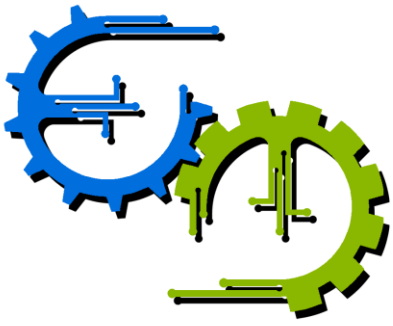
TRABALHO DE GRADUAÇÃO

SELEÇÃO DE COMPORTAMENTOS EM MÚLTIPLOS AGENTES AUTÔNOMOS COM APRENDIZAGEM POR REFORÇO EM AMBIENTES ESTOCÁSTICOS

Por,

Matheus Vieira Portela

Brasília, dezembro de 2015



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

TRABALHO DE GRADUAÇÃO

SELEÇÃO DE COMPORTAMENTOS EM MÚLTIPLOS AGENTES AUTÔNOMOS COM APRENDIZAGEM POR REFORÇO EM AMBIENTES ESTOCÁSTICOS

Matheus Vieira Portela

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

Banca Examinadora

Prof. Dr. Guilherme Novaes Ramos, CIC/UnB _____
Orientador

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

Prof. Dr. Edson Alves da Costa Júnior, FGA/UnB _____
Examinador interno

FICHA CATALOGRÁFICA

PORTELA, MATHEUS VIEIRA
Seleção de Comportamentos em Múltiplos Agentes Autônomos com Aprendizagem por Reforço em Ambientes Estocásticos,

[Distrito Federal] 2015.

vii, 53p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2015). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Inteligência artificial 2. Sistemas multiagentes 3. Programação bayesiana 4. Aprendizado por reforço 5. Caça-predador

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

PORTELA, M. V., (2015). Seleção de Comportamentos em Múltiplos Agentes Autônomos com Aprendizagem por Reforço em Ambientes Estocásticos. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$013, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 53p.

CESSÃO DE DIREITOS

AUTOR: Matheus Vieira Portela

TÍTULO DO TRABALHO DE GRADUAÇÃO: Seleção de Comportamentos em Múltiplos Agentes Autônomos com Aprendizagem por Reforço em Ambientes Estocásticos.

GRAU: Engenheiro ANO: 2015

Matheus Vieira Portela

SQN 214 Bloco J Apartamento 110 - Asa Norte.

70.873-100 Brasília, DF, Brasil.

Dedicatória

Aos meus pais, Ana Beatriz e Francisco Fábio, pelo esfoço constante para que eu sempre me dedicasse aos estudos.
Ao meu irmão, Marcelo, por estar sempre por perto quando preciso.
À minha amada, Ana Bárbara, pelo companheirismo sem igual e amor infinito.
Ao meu cachorro, Bandit, pelas recepções sempre calorosas ao chegar em casa.

Matheus Vieira Portela

Agradecimentos

Não há ser humano que consiga conquistar algo na vida sem contar com a ajuda daqueles que o rodeiam. Portanto, agradeço à minha família por todo apoio que sempre me foi dado para que pudesse dar o meu melhor nos estudos pois, sem vocês, jamais conseguiria chegar até aqui.

Agradeço também ao meu orientador, Prof. Dr. Guilherme Novaes Ramos, pela excelente orientação durante o desenvolvimento desse trabalho e por sempre cobrar para que eu fizesse o meu melhor. Certamente, evoluí muito academicamente graças à sua ajuda.

Pelas noites em claro e pelos desafios superados, agradeço aos amigos do time de futebol de robôs UnBall: Ícaro Mota, Gabriel Naves, Izabella Gomes, Gabriel Sousa, Vitor Duarte e Vieira Neto. Valeu a pena me divertir com vocês.

Agradeço aos meus amigos de projetos mirabolantes e boas risadas: Lucas de Levy, Tiago Pimentel, Guilherme Anselmo e George Brindeiro. Vocês me incentivam a evoluir pessoal e intelectualmente.

Também agradeço aos que acompanharam a jornada da Engenharia Mecatrônica comigo: Lucas Alcides, Felipe Ferreira, Nathalya Córdova, André Vargas, Uruatã Dias, Gabriel Teperino, Nicholas Travi, Narciane Muniz, Ana Clara da Hora e Allan Corrêa. Graças a vocês, sempre pude me divertir durante essa aventura.

Agradeço aos irmãos escoteiros: Márcio Cavalcanti, Raphael Tostes, Thaís Haluch, Glen Homer, André Porto, Matheus Araujo, Rodrigo Gregoldo, Bruno Gregoldo, Gláucio Magela, Anderson Takeshi, Alison Lopes, Mateus Melchior, Ivy Garcez, Jorge Santana, Bruna Louzada e Dayanna Rosa. Vocês me ensinaram que fazer o bem é sempre o correto.

Também agradeço aos amigos que fiz na Australian National University: Eduardo Neiva, Lucas Franco, Filipe dos Santos e Heitor Albuquerque. Com vocês, o intercâmbio na Austrália foi ainda melhor.

Agradeço a Nicole Maia, Débora Maia, Rafaela Dantas e Gabriela Andrea. Mesmo sendo difícil agendar um horário, sempre tivemos as melhores conversas possíveis.

Agradeço também a todos aqueles os quais não pude citar nominalmente, pois minha memória é limitada. Vocês fazem parte de quem eu sou.

E, por fim, agradeço à Ana Bárbara Ferrari, por sempre me apoiar, me incentivar e me fortalecer. Você é o amor da minha vida.

Matheus Vieira Portela

RESUMO

Agentes inteligentes agem baseados nas suas medições sensoriais a fim de alcançar seus objetivos. Em ambientes dinâmicos, como sistemas multiagentes, agentes devem adaptar seus processos de seleção de ações de acordo com o estado do sistema mutável, uma vez que comportamentos anteriormente considerados adequados podem tornar-se sub-ótimos. Tal problema é ainda maior se o ambiente é estocástico, forçando os agentes a lidarem com incertezas. Esse trabalho propõe um algoritmo de aprendizado por reforço para sistemas multiagentes estocásticos, utilizando programação bayesiana para estimação de estados e *Q-learning* com aproximação de funções para prover aos agentes a capacidade de aprender a selecionar os comportamentos mais adequados. Experimentos indicam resultados positivos para a abordagem, onde agentes aprenderam a cooperar, de forma autônoma, em um jogo eletrônico estocástico multiagente.

Palavras-chave: Inteligência artificial, sistemas multiagentes, programação bayesiana, aprendizado por reforço, caça-predador.

ABSTRACT

Intelligent agents act based on sensor measurements in order to fulfill their goals. When the environment is dynamic, such as a multiagent system, agents must adapt their action selection processes according to the changes in the system's state, given that behaviors that previously were considered the best choice may becomes sub-optimal. This problem is even greater when the environment is stochastic, forcing the agents to deal with uncertainties. This work proposes a reinforcement learning algorithm for stochastic multiagent systems, using Bayesian programming for state estimation and Q-learning with function approximation to provide the agents with capabilities to select the most appropriate behaviors. The experiments indicate positive results for this approach, where agents autonomously learned to cooperate in a stochastic multiagent digital game.

Keywords: Artificial intelligence; multiagent systems; Bayesian programming; reinforcement learning; predator-pursuit;

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

$\mathcal{G}$	Distribuição gaussiana
N	Número de agentes
Q	Valor do par estado-ação ou estado-comportamento
r	Recompensa
S	Estado do sistema
t	Tempo
U	Ação do agente
V	Valor do estado
Z	Medição sensorial

Símbolos Gregos

α	Fator de aprendizado
γ	Fator de desconto
δ	Dados coletados
ϵ	Ruído de medição
μ	Média
θ	Parâmetro
π	Conhecimentos do projetista
σ	Desvio-padrão
ϕ	Característica

Subscritos

d	Estado ou medição de distância
i	Identificação do agente
θ	Estado ou medição de direção

Sobrescritos

t	Instante de tempo
$0:t$	Intervalo de tempo

Siglas

IA	Inteligência Artificial

Capítulo 1

Introdução

"Se queres prever o futuro, estuda o passado." – Confúcio

Em meados do século XX, diversos cientistas dedicavam seus esforços para o estudo de máquinas que pudessem agir de forma autônoma. Até a primeira metade do século, a teoria de controle avançava a passos largos, incorporando os mais recentes avanços da matemática, física, elétrica e eletrônica para controlar sistemas mecânicos [1].

Foi durante a Segunda Guerra Mundial que Norbert Wiener, estudioso da teoria de controle, começou a criar sistemas eletromecânicos que imitassem comportamentos de seres biológicos [2]. Seus estudos fundaram a *cibernética*, que combina teorias da engenharia com princípios da biologia e da neurociência [2]. Essa pesquisa influenciou diretamente W. Grey Walter que, por sua vez, criou uma série de tartarugas eletromecânicas, as quais são consideradas os primeiros robôs e as primeiras formas artificiais de vida criados pela humanidade [3].

Até então, as pesquisas com formas artificiais de vida envolviam a interação física dos robôs com o ambiente, bem como as arquiteturas de suas inteligências [1]. Poucos anos depois, em 1956, a Inteligência Artificial (IA) toma forma como área de estudo cujo objetivo é criar inteligência computacional com capacidades semelhares às faculdades humanas, possuindo habilidades como criatividade, capacidade de adaptação e uso de linguagem [4].

Assim surgiu a robótica, agregando elementos da teoria de controle, da cibernética e da IA. Atualmente, a robótica trata do estudo de sistemas autônomos que existem no mundo físico, percebem o ambiente no qual estão imersos e agem a fim de alcançar seus objetivos [1].

1.1 Robótica probabilística

Ao observar a robótica por meio da definição anterior, percebe-se porque essa área de estudo é tão complexa. O mundo físico é inerentemente estocástico, uma vez que modelos matemáticos descrevem a natureza de forma simplificada. Sensores são inerentemente imprecisos nas suas medições e limitados por suas características físicas. Atuadores também são imprevisíveis devido a ruídos de controle e falhas mecânicas. Por fim, incertezas surgem no *software* dos robôs por meio das formas limitadas de representação de conhecimentos acerca do ambiente [5].

Para lidar com tantas incertezas nos mais diversos componentes da robótica, a teoria da probabilidade e os métodos estatísticos vêm sendo utilizados em aplicações acadêmicas e industriais com resultados surpreendentes: mapeamento de ambientes desconhecidos [6], reconhecimento de cenas utilizando câmeras [7] e robôs que jogam futebol [8] são alguns dos casos de sucesso.

Em muitas aplicações robóticas, é relativamente simples decidir como agir quando o estado do ambiente é completamente conhecido. Entretanto, usualmente, um robô não é capaz de conhecer o estado real do sistema, uma vez que a única forma de adquirir conhecimento acerca do ambiente é utilizando os seus sensores, os quais não são exatos. Portanto, estimar o estado do sistema a partir de dados sensoriais é o alicerce de todos métodos da robótica probabilística [5].

Dentre os métodos de estimação propostos para lidar com as incertezas do ambiente de forma computacionalmente eficiente, destacam-se os métodos bayesianos, os quais utilizam inferência bayesiana para chegar a conclusões razoáveis mesmo em ambientes estocásticos. A programação bayesiana de robôs provê uma metodologia genérica para modelagem e tomada de decisões em robôs baseado somente em inferência bayesiana [9].

1.2 Sistemas multiagentes

Apesar de não haver um consenso na literatura, é necessário utilizar um definição de agentes para limitar o escopo desse trabalho. Dessa forma, considerar-se-á que um agente é algo que percebe um ambiente por meio de sensores e atua nesse ambiente, de forma autônoma, por meio de atuadores [4]. Mais especificamente, agentes racionais buscam maximizar uma determinada medida de desempenho, que é função do estado do ambiente, por meio de suas ações [10]. Dessa forma, um agente que executa ações escolhidas ao acaso não é considerado um agente racional.

Quando múltiplos agentes coexistem em um ambiente, ele é classificado como um sistema multiagente. Em sistemas multiagentes competitivos, os objetivos dos agentes são conflitantes e a maximização da medida de desempenho de um determinado agente influencia negativamente o desempenho de outros agentes [10]. Assim, os agentes precisam competir para que poucos, se não um único, consigam cumprir seus objetivos [4]. Por exemplo, em um jogo de xadrez, para que um agente vença a partida, necessariamente o outro agente precisa perdê-la.

Já em sistemas multiagentes cooperativos, agentes combinam seus esforços para atingir um objetivo em comum, permitindo que os objetivos de múltiplos agentes sejam cumpridos simultaneamente [10]. É o caso de múltiplos robôs vasculhando uma área para encontrar um fugitivo, pois, quando o sujeito for localizado, todos os agentes alcançarão seus objetivos. É importante ressaltar que um ambiente pode conter, ao mesmo tempo, algumas características competitivas e outras cooperativas, portanto, tais definições não são mutuamente excludentes [4].

1.3 Aprendizado

Em ambientes dinâmicos, como sistemas multiagentes, um comportamento considerado ótimo em um momento pode deixar de sê-lo em um momento posterior. Dessa forma, um agente racional, que busca maximizar seu desempenho, precisa de mecanismos de aprendizado para adaptar-se às mudanças do ambiente. Aprendizado, neste trabalho, é entendido como a capacidade do agente de melhorar seu desempenho utilizando a experiência adquirida ao longo do tempo [4].

Entretanto, desenvolver máquinas que aprendem é um dos maiores desafios da inteligência artificial, grande suficiente para gerar uma área de estudo independente: aprendizado de máquinas [11]. Atualmente, três abordagens para aprendizado são as mais comuns: aprendizado supervisionado, aprendizado não-supervisionado e aprendizado por reforço [12].

Em aprendizado supervisionado, disponibiliza-se exemplos contendo dados de entrada e os dados de saída esperados, cabendo ao agente adaptar-se para conseguir estimar corretamente a saída para novos dados [13].

Aprendizado não-supervisionado, por sua vez, busca agrupar dados que sejam semelhantes de acordo com critérios aprendidos pelo agente ou determinar a distribuição de probabilidade que gerou tais dados [14].

Em aprendizado por reforço, técnica de aprendizado viável em sistemas multiagentes, o agente aprende quais ações devem ser executadas de forma a maximizar as recompensas numéricas recebidas do ambiente [10]. Nesse cenário, o agente não possui informações de quais ações são as melhores, tendo de descobri-las por conta própria ao interagir com o ambiente [13].

A maior parte dos algoritmos de aprendizado foram desenvolvidos com foco em sistemas com um único agente [15]. No entanto, sistemas multiagentes possuem características que tornam o aprendizado mais complexo, como espaços de estados particularmente grandes, função de recompensa variável de acordo com ações de outros agentes ou adaptação dos outros agentes [12]. Por não haver garantias de que algoritmos de aprendizado desenvolvidos em sistemas de único agente funcionam em sistemas multiagentes, é interessante estudar os algoritmos que possibilitam o aprendizado nesses sistemas.

1.4 Comportamentos de navegação

Ao desenvolver agentes racionais, deve-se determinar quais são as ações passíveis de serem executadas de acordo com as suas limitações físicas. Por exemplo, carros podem ir para frente, para trás e fazer curvas, enquanto movimentos puramente laterais são impossíveis. Dessa forma, algoritmos de aprendizado mapeiam estados para ações, selecionando a melhor ação estimada para ser executada no estado atual do ambiente.

Entretanto, outra abordagem lida com agentes autônomos que são desenvolvidos em termos de comportamentos a serem executados. Para esse trabalho, define-se comportamento como o conjunto de ações realizados por um agente de acordo com o estado do ambiente [16]. Tal abordagem

é especialmente significativa quando utilizadas em algoritmos de aprendizado, os quais aprendem a selecionar comportamentos ao invés de ações primitivas, reduzindo o espaço de estados e acelerando o aprendizado, o que torna os algoritmos mais eficientes [17].

Dentre os estudos realizados acerca de comportamentos para agentes autônomos, destacam-se os comportamentos de navegação. Investigados desde as origens da robótica [3], comportamentos de navegação são um conjunto de comportamentos – desde os mais simples até outros complexos – que visam imitar os comportamentos observados no mundo real, em especial, aqueles gerados por animais, como pássaros voando em formação [18].

1.5 Problemas de caça-predador

A validação de algoritmos de IA é comumente realizada em cenários conhecidos pela comunidade científica, como o problema de caça-predador. Esse cenário é frequentemente aplicado em algoritmos para sistemas multiagentes devido à facilidade de configurá-lo de forma a testar diferentes características do algoritmo em questão [19].

Apesar de não possuir um único formato, o problema de caça-predador tradicionalmente possui, em um ambiente discreto, uma caça e quatro predadores [19]. A caça deve evitar ser capturada, enquanto os predadores visam capturá-la. Como o objetivo dos predadores é comum, esse cenário é caracterizado como um sistema multiagente cooperativo [19].

Dentre os trabalhos existentes na literatura utilizando problemas de caça-predador, destacam-se a evolução de estratégias de múltiplos agentes com algoritmos genéticos [20], criação de estratégias de busca levando em consideração limitações físicas dos sensores [21] e estudos em robótica coevolucionária [22, 23].

É possível ainda destacar o interesse em problemas de caça-predador fora do âmbito acadêmico. Em 2010, a Agência de Projetos de Pesquisa Avançada de Defesa[1] dos Estados Unidos da América iniciou o projeto ACTUV[2] cujo objetivo é construir veículos marítimos autônomos para localizar e interceptar submarinos em ambientes hostis[3]. Considerando o submarino inimigo como a caça, múltiplos ACTUVs desempenhariam o papel de predador, os quais beneficiariam-se ao trabalhar de forma cooperativa.

1.6 Caracterização do problema

Considerando o contexto apresentado, surge a seguinte pergunta norteadora: é possível que múltiplos agentes, em um ambiente estocástico, aprendam comportamentos cooperativos? Na literatura, trabalhos acerca de aprendizado em sistemas multiagentes foram desenvolvidos utilizando teoria dos jogos, embora geralmente para sistemas multiagentes competitivos [24, 25]. Abordagens

[1] Em inglês, *Defense Advanced Research Projects Agency* ou DARPA.

[2] Em inglês, *Anti-Submarine Warfare Continuous Trail Unmanned Vessel*.

[3] Mais informações em: http://www.darpa.mil/program/anti-submarine-warfare-continuous-trail-unmanned-vessel.

comuns para sistemas multiagentes cooperativos envolvem o uso de algoritmos genéticos [12], os quais são considerados heurísticas de busca e não incorporam o aprendizado durante a execução do sistema.

Entretanto, a pergunta norteadora possui um escopo muito abrangente, sendo extremamente difícil respondê-la em apenas um trabalho de pesquisa. Portanto, decidiu-se analisá-la no contexto de um caso específico de sistema multiagente, com problemas de caça-predador. Mais além, utilizou-se programação bayesiana para lidar com as incertezas de um sistema estocástico e *Q-learning* com aproximação de funções para prover capacidade de aprendizado para os agentes. O estudo de *Q-learning* e da programação bayesiana em sistemas multiagentes já foi realizado na literatura, embora não em conjunto. Ao unir ambas as teorias, é possível obter uma abordagem genérica para controlar agentes computacionais e robôs em quaisquer ambientes estocásticos, em princípio, necessitando de pouca ou nenhuma adaptação do algoritmo. Dessa forma, este trabalho busca responder à pergunta específica: é possível que múltiplos agentes, em problemas de caça-predador, aprendam comportamentos cooperativos utilizando programação bayesiana e *Q-learning* com aproximação de funções?

Em robótica, a utilização de simuladores para validação inicial de algoritmos é tão comum que diversos simuladores de robótica foram desenvolvidos nas últimas décadas, tanto para uso científico quanto comercial [26]. Além de reduzir custos do projeto, uma vez que não é necessário alocar verbas para compra de robôs e manutenção devido a danos mecânicos, diminui-se tempo de desenvolvimento, uma vez que simulações computacionais costumam serem mais rápidas que testes físicos [27].

Jogos eletrônicos, por sua vez, também são utilizados como simuladores para validação de algoritmos de robótica [28]. Atualmente, certos jogos são utilizados para desenvolver algoritmos de IA com aplicação na robótica. Em especial, destaca-se a competição mundial de futebol de robôs, RoboCup[4], que possui a categoria *Soccer Simulation League*[5] na qual os participantes competem com algoritmos nos simuladores em 2D ou 3D. A metodologia desenvolvida nesse trabalho é testada em simulações computacionais de problemas de caça-predador utilizando o jogo *Pac-Man*.

No Capítulo 2, descreve-se os fundamentos da programação bayesiana e do aprendizado por reforço, bem como o algoritmo *Q-learning* e sua extensão com aproximação de funções, além de comportamentos de navegação. No Capítulo 3, desenvolve-se uma proposta de algoritmo de aprendizado para um sistema estocástico multiagente cooperativo, além de descrever os experimentos que serão realizados para sua validação. O Capítulo 4 apresenta os resultados coletados por meio dos experimentos, bem como suas análises. Por fim, o Capítulo 5 encerra o texto com as conclusões alcançadas e trabalhos futuros.

[4]Acessível em: http://www.robocup2015.org/

[5]Mais informações em: http://wiki.robocup.org/wiki/Soccer_Simulation_League

Capítulo 2

Fundamentação Teórica

"Lembre-se, os grandes segredos estão contidos nas técnicas básicas." – Morihei Ueshiba

Neste capítulo, apresenta-se a teoria de comportamentos de navegação. Em seguida, discute-se acerca dos fundamentos da aprendizagem por reforço e, então, apresenta-se o algoritmo de aprendizado por reforço *Q-learning* e a sua extensão com aproximação de funções. Por fim, discorre-se sobre a teoria matemática que embasa a programação bayesiana.

2.1 Comportamentos de navegação

Os primeiros estudos sobre comportamentos precedem, em muito, a robótica e mesmo a computação. De fato, é possível encontrar estudos sobre comportamentos animais desde as investigações realizadas pelo britânico Charles Darwin, em meados do século XIX, quando o próprio relata:

> "O [pombo] cambalhota comum possui o hábito singular e inteiramente hereditário de voar em bandos compactos a grandes alturas, dando eventualmente uma cambalhota completa em pleno vôo." [29]

Os estudos sobre a teoria da evolução de Darwin influenciaram diretamente a etologia, a ciência que estuda o comportamento animal [30]. O comportamento humano, no entanto, não é o principal objeto de estudo da etologia, razão pela qual surge tanto a psicologia[1] quanto a sociologia [31]. De forma simplista, enquanto a primeira costuma analisar indivíduos, a segunda concentra-se no comportamento social de grupos humanos.

Devido a presença de comportamentos nas mais diversas áreas de conhecimento, é natural que robôs sejam desenvolvidos para reproduzir comportamentos esperados pelo seu projetista ao invés de executar unicamente ações primitivas. Quando W. Grey Walter desenvolveu suas conhecidas tartarugas eletromecânicas, o fez para que fossem atraídas para a luz e repelidas pela sua ausência. Mais ainda, sua *Machina docilis* possuía capacidade de aprendizado. Após cerca de 20 tentativas, a

[1] Segundo a *American Psychological Association*, psicólogos estudam estados mentais normais e anormais, bem como comportamentos. Fonte: http://www.apa.org/about/index.aspx

máquina conseguia associar um sinal sonoro com a presença de luz, tal qual um rato de laboratório treinado para buscar comida [3].

Ao longo do desenvolvimento da robótica, percebeu-se que comportamentos complexos poderiam surgir pela composição de outros mais simples, sem necessitar do planejamento de todas as ações necessárias para atingir determinado estado. Desenvolveu-se, portanto, arquiteturas reativas ou arquiteturas baseadas em comportamentos, nas quais comportamentos eram ativados de acordo com a combinação das medições sensoriais [32].

Dentre os métodos de composição de comportamentos propostos, dois tornaram-se mais populares: supressão e campos de potencial. A composição por supressão permite ativar ou desativar comportamentos de acordo com as leituras sensoriais. Já a composição por campos de potencial busca somar as reações de cada comportamento para gerar uma ação resultante intermediária [1].

Na década de 1990, os estudos de Reynolds, que visavam a criação de comportamentos para agentes autônomos virtuais capazes de imitar comportamentos observados no mundo real, culminaram nos denominados comportamentos de navegação [18]. Assim, apresentam-se comportamentos simples cujas ações resultantes são influenciadas pelo posicionamento dos agentes no ambiente e que, quando combinados, tornavam cada vez mais complexa e natural a locomoção dos agentes.

A teoria de comportamentos de navegação assume um modelo bidimensional de veículo que se move por um ambiente [18], o qual pode ser controlado pela sua aceleração, frenagem e direção. Nessa teoria, a direção do veículo é a principal forma de controle estudada, sendo controlada para alinhar-se a um vetor de direção, cujo ângulo é calculado em função do posicionamento de outros agentes no ambiente. Assim, os comportamentos de navegação obtidos são definidos pela maneira com a qual esse vetor altera o seu ângulo e podem ser combinados por meio do somatório das suas forças resultantes [18].

Abaixo, detalham-se os comportamentos relevantes para esse trabalho de acordo com a teoria dos comportamentos de navegação de Reynolds [18].

2.1.1 Fuga

Sendo um dos comportamentos mais fáceis de identificar em animais, o comportamento de fuga almeja afastar o agente dos seus predadores. Assim, calcula-se qual ação provavelmente deixará o agente em questão mais longe possível dos seus adversários, conforme a Figura 2.1.

2.1.2 Busca

O comportamento de busca possui caráter predatório, isto é, faz com que o agente siga, em linha reta, uma trajetória em direção a posição atual de outro determinado agente. A Figura 2.2 ilustra essa situação.

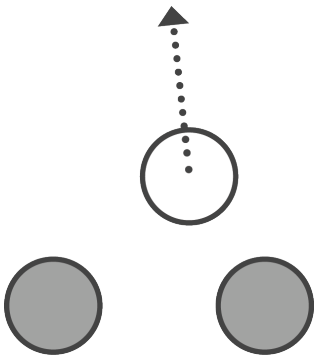


Figura 2.1: Representação do comportamento de fuga. O agente (em claro) movimenta-se conforme indicado a fim de afastar-se dos adversários (em escuro).

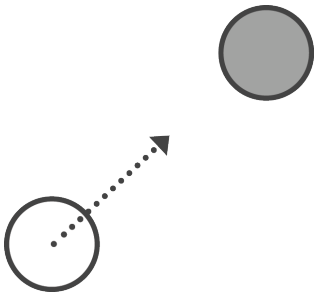


Figura 2.2: Representação do comportamento de busca. O agente (em claro) movimenta-se conforme indicado a fim de alcançar o adversário (em escuro).

2.1.3 Perseguição

Quando agentes encontram-se em movimento, o comportamento de busca pode apresentar resultados insatisfatórios, uma vez que o agente pode não conseguir capturar sua caça em tempo finito. Por exemplo, quando os agentes estão se movendo na mesma direção e com a mesma velocidade, a distância relativa entre os agentes mantém-se constante, portanto, a captura nunca será bem-sucedida.

O comportamento de perseguição surge para resolver tal questão. Esse comportamento move o agente de forma a alcançar a posição futura da caça, leva em consideração o deslocamento atual do seu inimigo, como apresenta a Figura 2.3.

2.2 Aprendizado por reforço

Uma vez que o agente tenha definido um conjunto de comportamentos que podem ser utilizados para locomoção no espaço, surge a pergunta: para determinado estado do ambiente, qual comportamento o agente deve selecionar para conseguir cumprir seus objetivos?

A ideia de aprender a selecionar comportamentos a partir de interações com o ambiente é natural ao ser humano, afinal, é por meio de tentativa e erro que desenvolvem-se os sistemas locomotor, fonético e cognitivo. Frequentemente, não há uma classificação explícita sobre o que é

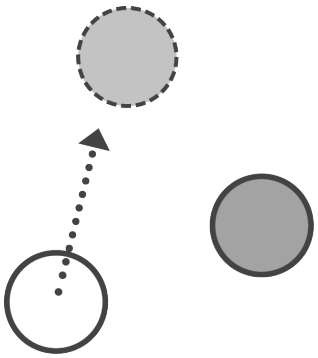


Figura 2.3: Representação do comportamento de perseguição. O agente (em claro) movimenta-se levando em consideração a posição futura (em tracejado) do adversário (em escuro).

certo ou errado, mas apenas uma indicação de que uma ação levou o agente para mais próximo ou mais distante do seu objetivo [13].

Para IA, aprendizado por reforço representa uma categoria de técnicas computacionais para que agentes melhorem seu desempenho ao longo do tempo a partir das suas interações com o ambiente no qual está inserido. A abordagem utilizada nesse trabalho baseia-se na teoria desenvolvida por Sutton e Barto [13].

2.2.1 Elementos de aprendizagem por reforço

Um agente deve ser capaz de perceber o estado s do ambiente no qual está inserido, executar ações a que modifiquem o ambiente e possuir objetivos [13]. O aprendizado é importante quando o agente não possui, *a priori*, o conhecimento de qual ação executar em cada possível estado do sistema de forma a atingir seu objetivo eficientemente. Diferentemente de aprendizado supervisionado, no qual o agente aprende a partir de exemplos dados por uma entidade externa, em aprendizado por reforço o agente deve aprender a partir de suas próprias experiências ao interagir com o ambiente [13].

Nesse contexto, define-se como sinal de recompensa r um valor numérico disponibilizado ao agente pela função de recompensa $R(s)$ ou $R(s, a)$, a qual avalia a eficiência do agente em atingir o seus objetivos [13]. Assim, o agente busca executar ações que maximizem as recompensas recebidas ao longo da sua execução. Por exemplo, um predador que recebe recompensa $+100$ quando capturar a presa e -1 enquanto ela não for alcançada busca se comportar de forma a maximizar a quantidade capturas realizadas. Por outro lado, a recompensa negativa faz com que o predador evite demorar para capturar a presa.

É importante notar que as recompensas podem ser recebidas imediatamente após a execução das ações ou após um determinado tempo. Na primeira situação, trata-se de um problema de tentativa e erro [13]. Já na segunda situação, há um problema de aprendizado por reforço com recompensas atrasadas [13].

Relacionado à recompensa, define-se valor V como sendo a quantidade de recompensas que um

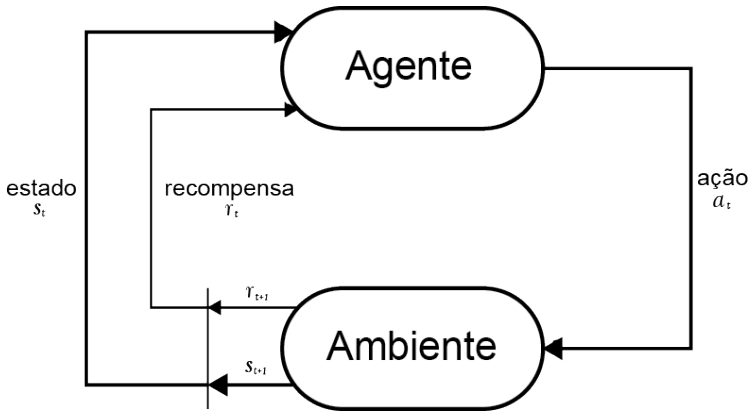


Figura 2.4: Diagrama de aprendizado por reforço.

agente espera acumular ao longo do tempo a partir do seu estado atual [13]. Qualitativamente, a função de valor $V(s)$ avalia o desempenho do agente a longo prazo, enquanto a função de recompensa avalia o desempenho de curto prazo. No exemplo anterior, um predador que está a duas células de distância de uma presa irá receber recompensa -1 no primeiro passo e $+100$ no segundo. Embora a recompensa imediata seja negativa, o agente foi capaz de executar tal ação visando a captura da presa, que futuramente gerou uma recompensa positiva significativa.

É perceptível que as decisões do agente devem ser tomadas levando em consideração valores ao invés de recompensas, já que, muitas vezes, é necessário realizar sacrifícios imediatos para alcançar ganhos futuros maiores [13]. Contudo, estimar a função de valores não é uma tarefa trivial, sendo esse o maior desafio para algoritmos de aprendizado por reforço [13].

Diferentemente de programação bayesiana, em aprendizado por reforço π representa a política de um agente, isto é, um mapeamento de estado para ação que define a maneira pela qual o agente age ao longo do tempo [13]. Uma política pode ser implementada por tabelas, funções simples ou mesmo processos complexos de busca. Algoritmos de aprendizado por reforço nada mais são que métodos automatizados para que agentes aprendam políticas de forma autônoma [13]. Por exemplo, um predador pode aprender uma política agressiva para alcançar a caça da forma mais rápida possível. A caça, por sua vez, pode adotar uma política para fugir do predador andando em círculos.

Uma decisão importante no projeto de um agente com aprendizado por reforço é a sua forma de aprendizado: definir o quanto o agente deve explorar, isto é, executar ações consideradas não-ótimas, a fim de receber recompensas maiores no futuro. Como exemplo, um predador aprende a capturar a caça ao andar sempre em círculos no centro do ambiente. Nesse caso o comportamento de andar em círculos possui é estimado com elevado V por receber recompensas positivas ao capturar a caça. Embora esse comportamento seja considerado o melhor naquele momento, o predador pode decidir adotar ocasionalmente outros comportamentos com V menor para conferir se há outra forma mais eficiente de alcançar seu objetivo que ainda não foi testada. Eventualmente, esse predador pode descobrir que a caça é capturada mais rapidamente caso adote o comportamento de mover-se diretamente em direção à sua presa, o que irá aumentar a estimativa de V para esse comportamento.

O aprendizado guloso seleciona sempre a ação que irá levar o agente para um estado com maior valor V de acordo com as estimativas atuais do agente [13]. Já no aprendizado ϵ-guloso, há uma chance ϵ que o agente selecione uma ação aleatória para fins de exploração e $1 - \epsilon$ que o agente escolha a ação com maior valor V [13].

Por fim, problemas de aprendizado por reforço podem utilizar um modelo do ambiente, algo que simule o comportamento do ambiente. Tais modelos são usados no planejamento de sequências de ações a serem tomadas levando em consideração os estados futuros que serão alcançados.

2.2.2 Aprendizado com diferença temporal

O valor V é utilizado para selecionar as ações e comportamentos que geram maiores recompensas para o agente. Entretanto, os valores V para os estados do ambiente são inicialmente desconhecidos, sendo necessário que os agentes utilizem métodos para aprendê-los.

O método de aprendizado por diferença temporal (em inglês, *temporal difference* ou TD) estima V a partir das recompensas obtidas ao executar ações. De maneira simples, TD pode ser compreendido como um método para corrigir a estimativa atual e aproximar-se dos valores recebidos na próxima vez [33]. A atualização é realizada imediatamente ao receber a recompensa em vez de ocorrer apenas quando alcançado o objetivo. Na Equação (2.1), tem-se o método mais simples de TD, conhecido como TD(0) [13],

$$V(s_{t-1}) \leftarrow V(s_{t-1}) + \alpha[r_t + \gamma V(s_t) - V(s_{t-1})] \tag{2.1}$$

onde s_{t-1} representa o estado do agente no instante anterior e s_t é o estado do agente no instante atual após executar uma ação a e receber, do ambiente, a recompensa r_t correspondente. A correção da estimativa do valor $V(s_{t-1})$ é realizada imediatamente ao receber tal recompensa.

Na Equação (2.1), α é o fator de aprendizado, com $0 < \alpha < 1$. Seu papel é limitar o passo de aprendizagem, ou seja, a variação máxima que $V(s_{t-1})$ irá sofrer em uma única iteração [13]. Mais além, γ é o fator de desconto, limitado ao intervalo $0 \leq \gamma \leq 1$, que permite dar maior relevância para recompensas imediatas ou futuras [13]. Quando γ aproxima-se de 0, o agente busca maximizar recompensas a curto prazo, enquanto recompensas a longo prazo são valorizadas com γ mais próximo a 1.

Para exemplificar o algoritmo TD(0), considere uma caça que está no estado x a uma célula de distância de um predador. Ao iniciar o processo de aprendizagem, a caça não possui informação acerca das recompensas que podem ser recebidas a partir desse estado, logo $V(x) = 0$. No estado seguinte y, ao ser capturada pelo predador, a caça recebe uma recompensa negativa $r_1 = -100$ e, a partir de tal informação, atualiza a estimativa de $V(x)$ conforme a Equação (2.1). Considerando que também não havia informações iniciais acerca do valor do novo estado do sistema, isto é, $V(y) = 0$, utilizando $\alpha = 0,5$ e $\gamma = 0,5$, a nova estimativa do valor do estado x é $V(x) = -50$, conforme apresentado na Equação (2.2). Dessa forma, a caça agora possui a informação que encontrar-se a uma célula de distância do predador é um estado que deve ser evitado, uma vez que $V(x) < 0$.

$$V(x) \leftarrow V(x) + \alpha[r_1 + \gamma V(y) - V(x)] = 0 + 0,5(-100 + 0,5 \cdot 0 - 0) = -50 \tag{2.2}$$

O TD é um algoritmo que estima o valor V de cada estado do ambiente, entretanto, sem especificar quais ações devem ser selecionadas para maximizar as recompensas futuras. Uma vez que o interesse desse trabalho é utilizar aprendizado por reforço para controlar agentes em um ambiente, é necessário utilizar métodos de aprendizado que permitam selecionar ações ou comportamentos adequados ao estado atual. O algoritmo de *Q-learning*, por exemplo, atende a essa necessidade.

2.2.3 Q-learning

O algoritmo *Q-learning* foi desenvolvido como algoritmo de controle de agentes baseado TD que, ao invés de estimar o valor do estado atual $V(s_t)$, estima valores de pares estado-ação $Q(s_t, a_t)$ [13]. Assim, um agente não armazena a informação de quanta recompensa pode ser recebida a partir de um estado, mas sim a recompensa que poderá ser recebida se, em um determinado estado s_t, o agente executar a ação a_t.

A sua forma mais simples, *Q-learning* com um passo, é apresentada na Equação (2.3) [13]. É interessante notar que o *Q-learning* não necessita do modelo matemático do ambiente no qual o agente está inserido, utilizando apenas as relações entre estado e ação [13]. Por esse motivo, um agente que implementa *Q-learning* como algoritmo de aprendizado não precisa modificá-lo para cada ambiente no qual for inserido.

$$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \alpha_t \left[r_t + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1}) \right] \tag{2.3}$$

Por exemplo, considere a situação na qual uma caça está no estado x a uma célula de distância de um predador e executa uma ação a' que a faz ser capturada. Assim, o valor de $Q(x, a')$ é atualizado incorporando a recompensa negativa recebida. Por utilizar *Q-learning*, a caça seleciona suas ações baseada nas estimativas $Q(s_t, a_t)$ selecionando as ações que possuem maior valor Q estimado para o estado atual. Na próxima vez que o sistema estiver no mesmo estado x, a caça evitará realizar a ação a' que causou sua captura, uma vez que $Q(x, a')$ é negativo, e poderá selecionar outra ação que a afaste do seu predador.

Matematicamente, é provado que as estimativas dos valores de pares estado-ação $Q(s_t, a_t)$ aprendidas utilizando *Q-learning* convergem para os valores ótimos e, portanto, possibilitando que o agente aprenda a política ótima para o seu ambiente [34]. Para tanto, é condição necessária que a correção da estimativa (2.3) seja executada por um longo período de tempo, isto é, $t \to \infty$, para que as estimativas dos valores de todos os estados do sistema sejam corrigidas inúmeras vezes até convergir ao valor real.

Mais ainda, para que a convergência do *Q-learning* ocorra, é necessário que o fator de aprendizado α_t respeite as condições descritas nas Equações (2.4) e (2.5) [34]. Um fator de aprendizado que atende a ambas as condições é descrito na Equação (2.6), onde K é um valor arbitrário escolhido pelo projetista e t indica a iteração atual do algoritmo.

$$\sum_{t=0}^{\infty} \alpha_t = \infty \tag{2.4}$$

$$\sum_{t=0}^{\infty} \alpha_t^2 < \infty \tag{2.5}$$

$$\alpha_t = \frac{K}{K+t} \tag{2.6}$$

2.2.4 Q-learning com aproximação de funções

Estimar os valores para cada par estado-ação é uma estratégia válida quando o ambiente é discreto, porém, torna-se problemática com espaços de estados contínuos, onde a quantidade de pares estado-ação é infinita [13]. Nessa situação, poucos estados serão visitados repetidamente, impedindo que as estimações dos valores dos pares estado-ação $Q(s_t, a_t)$ sejam atualizadas diversas vezes e, assim, podendo não convergir para os valores reais [13].

O algoritmo apresentado na Equação (2.3) também não permite realizar generalizações, ou seja, utilizar o que foi aprendido em um subconjunto do espaço de estados para estimar valores $Q(s_t, a_t)$ dos estados que não foram visitados anteriormente [13]. Assim, mesmo sistemas discretos com espaço de estados grande pode tornar inviável a execução do algoritmo devido a limitações de *hardware*, como tamanho de memória e poder computacional do processador.

Ambos os problemas podem ser resolvidos ao utilizar métodos de aproximação para estimar o valor $Q(s_t, a_t)$. Nesse caso, ao invés de memorizar os valores de todos os possíveis pares estados-ação, armazena-se somente um vetor de parâmetros $\boldsymbol{\theta}$, na forma apresentada na Equação (2.7), o qual é utilizado para calcular a estimativa atual do valor $Q(s_t, a_t)$ [13]. O aprendizado, portanto, é realizado ao atualizar as componentes θ_i do vetor de parâmetros utilizando as recompensas recebidas pelo agente [13],

$$\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_n]^T \tag{2.7}$$

onde $\theta_i \in \mathbb{R}$.

Dentre os métodos de aproximação existentes, uma escolha comum é o método de aproximação linear, ou *perceptron* [13]. Embora existam alternativas, como regressão linear ou redes neurais, a escolha da aproximação linear deve-se principalmente à sua simplicidade de implementação, reduzindo tempo de desenvolvimento, e baixo custo de execução, que não impacta o desempenho do algoritmo de aprendizado [35].

No método de aproximação linear, define-se um vetor de características $\boldsymbol{\phi}(s)$, tal qual a Equação (2.8) e de mesmo tamanho que $\boldsymbol{\theta}$, a partir do estado atual do ambiente. Dessa forma, o parâmetro θ_i define a influência que cada característica $\phi_i(s)$ terá no cálculo da aproximação de $Q(s, a)$,

$$\boldsymbol{\phi}(s) = [\phi_1(s), \phi_2(s), \ldots, \phi_n(s)]^T \tag{2.8}$$

onde $\phi_i(s) \in \mathbb{R}$, $0 \leq \phi_i(s) \leq 1$.

Portanto, a partir dos vetores $\boldsymbol{\theta}$ e $\boldsymbol{\phi}(s_t)$, a estimativa atual do valor do par estado-ação $Q(s_t, a_t)$ é calculada com a Equação (2.9).

$$Q(s_t, a_t) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s_t) = \sum_{i=1}^{N} \theta_i \phi_i(s_t) \tag{2.9}$$

Por exemplo, em um problema de caça-predador, o vetor de características pode ser composto por duas características, ou seja, $\boldsymbol{\phi}(s) = [\phi_1(s), \phi_2(s)]^T$. A primeira, $\phi_1(s)$, assume os valores 0 ou 1 e indica se o predador encontra-se a uma célula de distância da caça. A segunda, $\phi_2(s)$, representa a probabilidade do predador não capturar a caça no estado s do sistema. Um vetor de parâmetros com valores $\boldsymbol{\theta} = [-10, 100]^T$ indica que $\phi_1(s)$ afeta negativamente o valor $Q(s_t, a_t)$ do par estado-ação e $\phi_2(s)$ afeta positivamente. Considerando que o predador não esteja próximo a presa e que a chance dela ser capturada seja de 5%, então o vetor de características para o estado atual é $\boldsymbol{\phi}(s) = [0, 95\%]^T$ e a estimativa do valor do estado atual é $Q(s_t, a_t) = 0 \cdot (-10) + 0,95 \cdot 100 = 95$.

Na aproximação linear, o aprendizado ocorre ao atualizar $\boldsymbol{\theta}$ de acordo com a recompensa recebida pelo agente. Usualmente, utiliza-se o método de gradiente descendente para atualizar $\boldsymbol{\theta}$, conforme apresentado na Equação (2.10) [13],

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha_t \, \delta \, \nabla_{\boldsymbol{\theta}} Q(s_t, a_t) \\ \delta &= r_t + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1}) \\ \nabla_{\boldsymbol{\theta}} Q(s_t, a_t) &= \boldsymbol{\phi}(s_t) \end{aligned} \tag{2.10}$$

onde o fator de desconto γ, assim como no TD(0), deve estar no intervalo $0 \leq \gamma \leq 1$ e o fator de aprendizado α_t deve respeitar as condições do *Q-learning* descritas nas Equações (2.4) e (2.5).

Embora o algoritmo *Q-learning* possua garantia de convergência para a política ótima, ao aplicar aproximação de funções, tal garantia deixa de existir mesmo com α_t respeitando as Equações (2.4) e (2.5) [35]. Para evitar esse efeito, alguns cuidados devem ser tomados, como normalizar os valores mínimos e máximos de cada característica e utilizar valores para α_t que decaem com o tempo [35].

2.3 Programação bayesiana

Os algoritmos de aprendizado por reforço selecionam, para o estado atual do sistema, o comportamento mais adequado de acordo com o seu objetivo. No entanto, tais algoritmos necessitam que o estado seja uma variável conhecida, o que não é verdade quando o sistema é estocástico, como no caso do mundo real [5]. Nessas situação, é necessário utilizar métodos que lidem com as incertezas do ambiente para poder controlar o agente.

Em IA, processos de tomada de decisões que levem em consideração as incertezas de um ambiente estocástico vem sendo estudados há algumas décadas, inicialmente com redes bayesianas

e modelos gráficos. Desde então, abordagens bayesianas, que utilizam teoria da probabilidade e são capazes de analisar matematicamente as múltiplas possibilidades do estado do ambiente, vêm substituindo técnicas tradicionais de robótica, que assumem uma única possibilidade como sendo a real. Como resultado, robôs que implementam algoritmos bayesianos são mais robustos a incertezas e possuem os melhores resultados em tarefas complexas em ambientes estocásticos [5].

Contudo, arquiteturas robóticas costumam utilizar inferência bayesiana apenas em pontos específicos dos sistemas, como localização e mapeamento - tais quais técnicas de grade de ocupação, localização de Markov, filtro de partículas e filtro de Kalman - ou planejamento - destacando-se as abordagens de processos de decisão de Markov parcialmente observáveis e modelos ocultos de Markov [9]. Assim, as arquiteturas tornam-se um híbrido de métodos determinísticos com técnicas estatísticas, aumentando sua complexidade de implementação e manutenção.

Nesse cenário, a programação bayesiana, inicialmente desenvolvida por Olivier Lebeltel em sua tese de doutorado, é uma abordagem que permite criar arquiteturas robóticas utilizando somente inferência bayesiana [9]. Por utilizar apenas equações de probabilidade para programar robôs, é possível incorporar estocasticidade e incertezas do sistema de maneira mais simples do que em sistemas híbridos, onde uma parte é probabilística e outra, determinística [9]. Dessa forma, o formalismo apresentado pela programação bayesiana a torna uma abordagem genérica para programação de robôs, sendo até capaz de reinterpretar matematicamente técnicas probabilísticas clássicas, como redes bayesianas, modelos ocultos de Markov, filtros de Kalman, entre outras [9].

Descreve-se abaixo as definições mais importantes acerca da programação bayesiana de acordo com os trabalhos desenvolvidos por Lebeltel *et al.* [9] e Koike [36].

2.3.1 Definições e notação

A unidade fundamental da programação bayesiana é a proposição lógica, isto é, uma hipótese, denotada por uma letra minúscula como a e b, com significado preciso e que pode assumir um valor lógico verdadeiro, V, ou falso, F [9]. Por exemplo, em um problema de caça-predador, a proposição a pode representar a afirmação "a caça está próxima" e b, a afirmação "um predador está próximo".

Para lidar com incertezas, atribui-se probabilidades às proposições lógicas, isto é, um valor numérico no intervalo $[0, 1]$ que representa a chance de determinada proposição lógica possuir o valor assumido [9]. Por exemplo, $P(a = V) = P(a) = 0,9$ significa que a proposição a definida anteriormente possui 90% de chance de ser verdadeira, ou seja, há 90% de chance de haver uma caça próxima.

Ao aplicar operadores lógicos, definem-se novas proposições a partir de relações entre as proposições já estabelecidas [9]. O operador conjunção $a \wedge b$ é uma proposição indicando que a e b são verdadeiras simultaneamente [9]. Com as proposições exemplificadas anteriormente, $P(a \wedge b) = 0,1$ significa que a chance de haver uma caça próxima ao mesmo tempo em que há um predador próximo é de 10%. Já o operador disjunção $a \vee b$ denota a relação de pelo menos uma das proposições a ou b serem verdadeiras [9]. Por exemplo, $P(a \vee b) = 0,95$ indica que há 95% de chance de haver

uma presa ou um predador próximo. Por fim, o operador negação $\neg a$ define a proposição de a sendo falsa [9]. Assim, $P(\neg a) = 0,1$ afirma que há 10% de chance de não haver uma caça nas proximidades.

Toda proposição plausível é fruto de conhecimentos prévios do projetista [36]. Por exemplo, a afirmação "a caça está próxima ao lago" só é possível porque o projetista possui conhecimentos acerca da existência da caça e do lago no espaço, bem como uma métrica para distância e a própria definição de espaço em si. Esse tipo de informação é resumida como sendo π e, portanto, a probabilidade de toda e qualquer proposição depende da corretude de π, o que é representado matematicamente pela probabilidade condicional na forma $P(a|\pi)$ [9]. Similarmente, se a probabilidade de a depende também de outra proposição b, então a probabilidade é condicionada a conjunção de b e π, isto é, $P(a|b \wedge \pi)$ [9]. Nesse caso, se b representa "o predador está longe do lago", então $P(a|b \wedge \pi)$ indica a probabilidade da caça estar próxima considerando que o predador está longe do lago e todos os outros conhecimentos prévios do projetista ao elaborar o sistema.

O conceito de variável discreta é também fundamental na programação bayesiana. Uma variável discreta X é composta por um conjunto de proposições lógicas x_i mutuamente exclusivas e exaustivas, ou seja, $x_i \wedge x_j$ é falso para todo $i \neq j$ e pelo menos uma proposição lógica x_i deve ser verdadeira [9]. A cardinalidade dessa variável, $||X||$, representa a quantidade de proposições que a compõem. Também é possível avaliar a probabilidade de uma variável discreta X, ou seja, a probabilidade da conjunção das proposições x_i que a compõem, conforme apresentado na Equação (2.11) [9],

$$P(X|\pi) = P(x_1 \wedge x_2 \wedge \ldots \wedge x_n|\pi) \tag{2.11}$$

onde $n = ||X||$.

A conjunção de duas variáveis discretas, denotada por $X \wedge Y$, é definida pela conjunção das proposições que compõem ambas as variáveis $x_i \wedge y_i$, sendo a resultante também uma variável discreta [9].

2.3.2 Regras de inferência

Conhecendo as probabilidades de certas proposições, utilizam-se regras matemáticas para calcular probabilidades desconhecidas de outras proposições. Essas regras são denominadas regras de inferência e esse processo é conhecido como inferência bayesiana.

A regra de Bayes calcula a probabilidade de uma proposição x dado a probabilidade de outra proposição y conforme a Equação (2.12).

$$P(x|y) = \frac{P(y|x) \cdot P(x)}{P(y)} \tag{2.12}$$

A regra da conjunção, que deriva da regra de Bayes, permite calcular a probabilidade de uma conjunção de proposições, como apresentado na Equação (2.13) [9].

$$P(x \wedge y|\pi) = P(x|\pi) \cdot P(y|x \wedge \pi) = P(y|\pi) \cdot P(x|y \wedge \pi) \tag{2.13}$$

A regra da normalização, apresentada na Equação (2.14), define a relação entre as probabilidades de uma proposição x e sua negação $\neg x$ [9].

$$P(x|\pi) + P(\neg x|\pi) = 1 \tag{2.14}$$

É importante notar que ambas as regras são válidas para variáveis discretas. As Equações (2.15) e (2.16) apresentam a regra da conjunção e da normalização, respectivamente, para variáveis discretas [9].

$$P(X \wedge Y|\pi) = P(X|\pi) \cdot P(Y|X \wedge \pi) = P(Y|\pi) \cdot P(X|Y \wedge \pi) \tag{2.15}$$

$$P(X|\pi) + P(\neg X|\pi) = 1 \tag{2.16}$$

As regras da conjunção e da normalização são suficientes para realizar inferência bayesiana e, matematicamente, todas as regras de inferência derivam da aplicação destas. Apesar de não ser absolutamente necessária, apresenta-se a regra da marginalização na Equação (2.17) devido a sua utilidade na modelagem e utilização de programas bayesianos [36],

$$\sum_{X} P(X \wedge Y|\pi) = P(Y|\pi) \tag{2.17}$$

onde o lado esquerdo da equação representa o somatório das probabilidades para todos os possíveis valores das proposições de X em conjunção com Y.

2.3.3 Elementos de programas bayesianos

A programação bayesiana apresenta uma maneira formal de especificar uma família de distribuições de probabilidade, sobre variáveis discretas, a fim de controlar agentes para executar tarefas [36]. Dessa forma, um programa bayesiano é o conjunto de distribuições de probabilidade acerca das variáveis relevantes ao contexto do agente em conjunto com as variáveis sobre as quais deseja-se obter informações [9]. Todo programa bayesiano é composto por duas partes: descrição e pergunta.

2.3.3.1 Descrição

A descrição de um programa bayesiano é a coleção todas as informações, representadas por variáveis discretas, relevantes para o problema sobre o qual o agente age [9]. Matematicamente, esse conhecimento é representado pela distribuição de probabilidade conjunta de um conjunto de variáveis discretas $\{X_1, X_2, \ldots, X_n\}$ utilizando conhecimentos prévios (π) e dados experimentais

(δ), tal qual a Equação (2.18) [9]. Utilizando essa equação e regras de inferência, é possível calcular a probabilidade de variáveis necessárias pelo sistema por meio da pergunta de um programa bayesiano.

$$P(X_1 \wedge X_2 \wedge \ldots \wedge X_n | \delta \wedge \pi) \tag{2.18}$$

Em geral, o cálculo da distribuição conjunta como apresentado na Equação (2.18) é de difícil computação, sendo necessário reduzir tal complexidade por meio de decomposição da distribuição aplicando hipóteses de independência condicional [9]. Diz-se que duas variáveis X e Y são condicionalmente independentes dada uma variável Z se, e somente se, $P(X \wedge Y|Z) = P(X|Z) \cdot P(Y|Z)$. Assim, o cálculo será realizado aplicando regras de inferência nas distribuições mais simples. A descrição do programa estará completa ao definir-se as formas das distribuições simples. Por exemplo, $P(X_1|X_2 \wedge \delta \wedge \pi)$ pode assumir a forma de uma distribuição uniforme ou uma distribuição gaussiana em torno de um valor inferido a partir de δ.

2.3.3.2 Pergunta

A pergunta, em um programa bayesiano, é uma distribuição de probabilidade que permite calcular a probabilidade de uma variável de interesse do projetista [9]. Por exemplo, em um problema de caça-predador, elabora-se uma pergunta do tipo "qual é a probabilidade do predador estar perto da caça".

Para um programa descrito por $P(X_1 \wedge X_2 \wedge \ldots \wedge X_n | \delta \wedge \pi)$, define-se três variáveis discretas a partir da conjunção de $X_1 \wedge X_2 \wedge \ldots \wedge X_n$: I, variável sobre a qual deseja-se obter informações, C, variável sobre a qual as informações são conhecidas, e D, variáveis sobre as quais as informações são desconhecidas [9]. A pergunta é matematicamente descrita pela Equação (2.19) [9].

$$P(I|C \wedge \delta \wedge \pi) \tag{2.19}$$

Utilizando regras de inferência, é possível responder à pergunta utilizando a descrição do programa, como apresentado na Equação (2.20) [9].

$$P(I|C \wedge \delta \wedge \pi) = \frac{P(I \wedge C|\delta \wedge \pi)}{P(C|\delta \wedge \pi)} = \frac{\sum_D P(I \wedge C \wedge D|\delta \wedge \pi)}{\sum_{D,I} P(I \wedge C \wedge D|\delta \wedge \pi} \tag{2.20}$$

Por exemplo, um problema de caça-predador é descrito pelas variáveis X para "caça próxima ao predador", Y para "caça próxima à parede" e Z para "predador próximo à parede". Definindo $I = X$ e $C = Y \wedge Z$, a pergunta é definida pela probabilidade da caça estar próxima ao predador dado a probabilidade da caça e do predador estarem próximos à parede. Em seguida, essa estimação pode alimentar outras partes do sistema inteligente, tal qual seleção de ações.

Capítulo 3

Desenvolvimento

"Sonhe grande, pois ter sonhos grandes dão o mesmo trabalho dos sonhos pequenos." – Jorge Paulo Lemann

Neste capítulo, desenvolve-se um programa bayesiano capaz para estimar o estado do ambiente em um sistema multiagente. Logo após, apresenta-se o algoritmo *Q-learning* com aproximação de funções para seleção de comportamentos, além dos comportamentos de navegação utilizados.

3.1 Variáveis pertinentes

Tradicionalmente, problemas de robótica probabilística utilizam as variáveis: estado S, medição Z e ação U [5]. O estado é uma variável que armazena informações mensuráveis acerca de um agente e o ambiente no qual está inserido. Uma ação, quando realizada por um agente, altera o estado do ambiente no qual está inserido. Uma medição é uma estimativa, realizada por algum sensor, sobre o estado de determinada variável do ambiente.

Todas as variáveis relevantes no contexto atual alteram seus valores conforme o tempo $t = 0, 1, 2, \ldots$ avança. Tal dinâmica é denotada pelo sufixo de tempo, tal como $X^{0:t} = X^0 \wedge X^1 \wedge \ldots \wedge X^t$, o qual indica os valores da variável X no intervalo $[0, t]$.

É importante ressaltar que todos os agentes do sistema executam o mesmo algoritmo proposto. Assim, a descrição do programa bayesiano e do algoritmo de aprendizado é realizada sob a perspectiva de um único agente, identificado pelo índice 1, o qual observa os agentes $2, 3, \ldots, N$.

Para o sistema multiagente em questão, com N agentes, as formas das variáveis S e Z são definidas respectivamente de acordo com as Equações (3.1) e (3.2), onde d_i é a distância para o i-ésimo agente, θ_i é a orientação para o i-ésimo agente, d_{mi} é a distância medida para o i-ésimo agente e θ_{mi} é a orientação medida para o i-ésimo agente. A variável U diz respeito à ação realizada pelo agente que executa o algoritmo.

$$S = [d_1, \theta_1, d_2, \theta_2, \ldots, d_N, \theta_N] \tag{3.1}$$

$$Z = [d_{m1}, \theta_{m1}, d_{m2}, \theta_{m2}, \ldots, d_{mN}, \theta_{mN}] \tag{3.2}$$

3.2 Estimação de estado

Para realizar estimação do estado atual do sistema, utiliza-se um programa bayesiano conforme detalhado abaixo.

3.2.1 Descrição

A descrição do programa bayesiano reside na probabilidade conjunta (3.3), ou seja, a probabilidade das variáveis de estado, medição e ação no decorrer do tempo.

$$P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t} | \pi) \tag{3.3}$$

sobre a qual aplica-se a regra da conjunção, conforme apresentada na Equação (2.15), para poder explicitar a probabilidade conjunta das variáveis no instante atual, $S^t \wedge Z^t \wedge U^t$, em função da evolução das variáveis ao longo do tempo $S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1}$.

$$P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t} | \pi) = P(S^t \wedge Z^t \wedge U^t | S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1} \wedge \pi) P(S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1} | \pi) \tag{3.4}$$

Computar a Equação (3.4) não é uma tarefa trivial, uma vez que requer-se conhecimento de todo o histórico das variáveis ao longo do tempo. No entanto, assumir independência condicional entre algumas variáveis pode tornar tal computação viável, tal como considerar que o sistema é um processo markoviano.

Um processo estocástico é dito markoviano se possuir a propriedade de Markov, a qual define que a probabilidade conjunta das variáveis do sistema é condicionalmente independente do seu histórico entre 0 e $t-1$ caso seu valor em $t-1$ seja conhecido. Na prática, a propriedade de Markov permite que um agente capaz de estimar o estado no qual se encontra atualmente não precisa armazenar todos os estados anteriores do sistema, nem levá-los em consideração no cálculo da descrição (3.4). Matematicamente, tal propriedade é representada pela Equação (3.5), onde X representa o estado do sistema [5].

$$P(X^t | X^{0:t-1}) = P(X^t | X^{t-1}) \tag{3.5}$$

Levando a propriedade de Markov em consideração, é possível fatorar a descrição (3.4) e remover a dependência condicional entre a probabilidade conjunta das variáveis no instante atual e o histórico do sistema, conforme a Equação (3.6).

$$P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t} | \pi) = P(S^t \wedge Z^t \wedge U^t | S^{t-1} \wedge Z^{t-1} \wedge U^{t-1} \pi) P(S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1} | \pi) \tag{3.6}$$

A atualização da probabilidade conjunta do sistema, apresentada na Equação (3.6), pode ser interpretada como um processo iterativo que depende de apenas dois fatores, facilitando seu cálculo: a condição inicial $P(S^0 Z^0 U^0|\pi)$ e da equação de atualização $P(S^t \wedge Z^t \wedge U^t|S^{t-1} \wedge Z^{t-1} \wedge U^{t-1}\pi)$. Na Equação (3.7), explicita-se ambos os fatores.

$$P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t}|\pi) = P(S^0 \wedge Z^0 \wedge U^0|\pi) \prod_{j=1}^{t} P(S^j \wedge Z^j \wedge U^j|S^{j-1} \wedge Z^{j-1} \wedge U^{j-1}\pi) \quad (3.7)$$

A condição inicial $P(S^0 \wedge Z^0 \wedge U^0|\pi)$ representa o conhecimento que o projetista do sistema possui acerca do ambiente quando o programa inicia a sua execução. Quando nenhum conhecimento prévio existe, tal probabilidade assume a forma de uma distribuição uniforme [36].

A equação de atualização $P(S^t \wedge Z^t \wedge U^t|S^{t-1} \wedge Z^{t-1} \wedge U^{t-1}\pi)$, por sua vez, pode ser simplificada de acordo com o problema analisado e com a composição das variáveis S, Z e U. No sistema markoviano descrito, o estado atual depende do estado anterior e da última ação realizada; as medições e a ação atual dependem do estado presente do sistema [5].

$$P(S^t \wedge Z^t \wedge U^t|S^{t-1} \wedge Z^{t-1} \wedge U^{t-1}\pi) = P(S^t|S^{t-1} \wedge U^{t-1}\pi)P(Z^t|S^t\pi)P(U^t|S^t\pi) \quad (3.8)$$

A partir da Equação (3.7), a descrição do programa bayesiano estará completa ao definir as formas das equações do lado direito da Equação (3.8). Com essa descrição e com as regras de inferência, torna-se possível realizar estimativas acerca das variáveis do sistema.

3.2.1.1 Formas paramétricas

Com as variáveis d_i, θ_i, d_{mi}, θ_{mi} e U, considera-se as seguintes relações de dependência:

- d_i^t depende apenas de d_i^{t-1} e U^t;
- θ_i^t depende apenas de θ_i^{t-1} e U^t;
- d_{mi}^t depende apenas de d_i^t;
- θ_{mi}^t depende apenas de θ_i^t;
- U^t depende apenas do algoritmo de seleção de ações, portanto, é independente das outras variáveis.

Por meio de tais relações, é possível simplificar o lado direito da Equação (3.8) nas probabilidades condicionais do estado S^t, das medições Z^t e da ação U^t, como apresentado nas Equações (3.9), (3.10) e (3.11) respectivamente.

$$P(S^t|S^{t-1} \wedge U^{t-1} \wedge \pi) = P(d_i^t|d_i^{t-1} \wedge U^{t-1} \wedge \pi)P(\theta_i^t|\theta_i^{t-1} \wedge U^{t-1} \wedge \pi) \quad (3.9)$$

$$P(Z^t|S^t \wedge \pi) = P(d^t_{mi}|d^t_i \wedge \pi)P(\theta^t_{mi}|\theta^t_i \wedge \pi) \tag{3.10}$$

$$P(U^t|S^t \wedge \pi) = P(U^t|\pi) \tag{3.11}$$

Para finalizar a descrição do programa bayesiano, é necessário definir as formas das probabilidades $P(d^t_i|d^{t-1}_i \wedge U^{t-1} \wedge \pi)$, $P(\theta^t_i|\theta^{t-1}_i \wedge U^{t-1} \wedge \pi)$, $P(d^t_{mi}|d^t_i \wedge \pi)$, $P(\theta^t_{mi}|\theta^t_i \wedge \pi)$ e $P(U^t|\pi)$ conforme apresentado abaixo.

Supondo que o agente é levado do estado S^{t-1} para o estado S^t ao executar uma ação a, possuindo um modelo de movimentação, é possível calcular as coordenadas (x, y) do agente em questão no instante t relativas à posição no instante $t-1$. O mesmo procedimento pode ser realizado para calcular (x_i, y_i), a posição do i-ésimo agente no instante t. Nessa situação, a distância esperada do agente em questão para o i-ésimo agente é calculada pela Equação (3.12).

$$\bar{d} = \sqrt{(x_i - x)^2 + (y_i - y)^2} \tag{3.12}$$

Utilizando o mesmo procedimento, a orientação do i-ésimo agente é calculada por meio da Equação (3.13).

$$\bar{\theta} = \tan^{-1} \frac{y_i - y}{x_i - x} \tag{3.13}$$

No entanto, aplicações que possuem obstáculos requerem um modelo mais complexo da distância, uma vez que os agentes não são capazes de movimentar-se em linha reta até o seu objetivo. Nesses casos, grafos de acessibilidade oferecem uma maneira simples para lidar com a geometria do ambiente [37].

A probabilidade $P(d^t_i|d^{t-1}_i \wedge U^{t-1} \wedge \pi)$ pode ser descrita por uma distribuição gaussiana cuja média está centrada na distância esperada $\bar{d}$, como apresentado na Equação (3.14). Já o desvio-padrão σ da distribuição gaussiana indica a confiança de que o valor real de d^t_i que deseja-se estimar esteja próximo da média $\bar{d}$. Uma vez que $\bar{d}$ depende do modelo (3.12) e da ação U^{t-1} que foi executada, tanto a precisão do modelo quanto a confiabilidade dos atuadores influenciam diretamente o desvio-padrão. Assim, o valor σ_d a ser utilizado pode ser definido pelo projetista levando em consideração esses fatores ou estimado a partir de dados δ coletados ao executar o sistema.

$$P(d^t_i|d^{t-1}_i \wedge U^{t-1} \wedge \pi) = \mathcal{G}(d^{t-1}_i, U^{t-1}), \mu = \bar{d}, \sigma = \sigma_d \tag{3.14}$$

De forma análoga, $P(\theta^t_i|\theta^{t-1}_i \wedge U^{t-1} \wedge \pi)$ pode ser descrita por uma distribuição gaussiana, apresentada na Equação (3.15), com média em $\bar{\theta}$ e desvio-padrão σ_θ, influenciado pela qualidade do modelo (3.13) e dos atuadores.

$$P(\theta^t_i|\theta^{t-1}_i \wedge U^{t-1} \wedge \pi) = \mathcal{G}(\theta^{t-1}_i \wedge U^{t-1}), \mu = \bar{\theta}, \sigma = \sigma_\theta \tag{3.15}$$

A medição de distância do i-ésimo agente depende do estado atual, bem como da qualidade do sensor de distância, o qual influencia σ_{sd}. Portanto, assume a forma de uma distribuição gaussiana com média em d_i^t e desvio-padrão em σ_{sd}, como na Equação (3.16).

$$P(d_{mi}^t | d_i^t \wedge \pi) = \mathcal{G}(d_i^t), \mu = d_i^t, \sigma = \sigma_{sd} \tag{3.16}$$

Similarmente, a medição de orientação apresenta-se como uma distribuição gaussiana, tal qual a Equação (3.17), com média em θ_i^t e desvio-padrão $\sigma_{s\theta}$, influenciado pela qualidade do sensor de orientação.

$$P(\theta_{mi}^t | \theta_i^t \wedge \pi) = \mathcal{G}(\theta_i^t), \mu = \theta_i^t, \sigma = \sigma_{s\theta} \tag{3.17}$$

Por fim, a ação é selecionada pelo algoritmo de seleção de ação, que envia ações diretamente para o agente. Como U^t não é conhecido pelo programa bayesiano, representa-se $P(U_i^t|\pi)$ por uma distribuição uniforme [36].

É necessário ressaltar que outras distribuições de probabilidade, além da distribuição gaussiana, podem ser utilizadas para definir as formas paramétricas dependendo da aplicação. Por exemplo, a distribuição de Rayleigh é mais adequada ao modelar sinais eletromagnéticos em sistemas de comunicação cuja potência decai de acordo com a distância entre receptor e transmissor [38].

3.2.2 Pergunta

Com a descrição do programa bayesiano em mãos, é possível utilizá-lo para formular uma pergunta probabilística. Em outras palavras, utilizando a descrição do sistema (3.7), calcula-se estimativa acerca de variáveis do sistema.

No contexto apresentado nesse trabalho, deseja-se obter a estimação do estado atual do sistema dadas as medições mais recentes e as ações executadas no instante anterior, uma vez que tal informação é necessária para a seleção de comportamento. A pergunta, portanto, assume a forma matemática da Equação (3.18).

$$P(S^t | Z^t \wedge U^{t-1} \wedge \pi) \tag{3.18}$$

Conforme apresentado na Equação (2.20), é possível calcular a pergunta a partir da descrição por meio de inferência bayesiana. Nesse algoritmo, $Z^t \wedge U^t$ é a variável conhecida e S^t representa a variável procurada, não existindo variável desconhecida. Assim, a pergunta é calculada por meio da Equação (3.19).

$$P(S^t | Z^t \wedge U^{t-1} \wedge \pi) = \frac{P(S^t \wedge Z^t \wedge U^t | \pi)}{\sum_{S^t} P(S^t \wedge Z^t \wedge U^t | \pi)} \tag{3.19}$$

A Equação (3.19) representa, portanto, uma maneira sistemática de estimar a probabilidade do sistema encontrar-se em um determinado estado utilizando as medições sensoriais atuais e a última

ação realizada. A partir dessa informação, torna-se possível calcular qual é o estado do sistema mais provável, informação que pode, então, alimentar outros sistemas do agente que necessitam do conhecimento do estado atual para poder atuar.

3.3 Seleção de comportamento

O algoritmo *Q-learning* tradicional visa estimar o valor dos pares estado-ação para, então, poder selecionar ações que possuam os maiores valores estimados $Q(s_t, a_t)$ para o estado atual. Algumas das ações possíveis de serem realizadas em um problema de caça-predador incluem mover-se para norte, para sul, para leste, para oeste ou permanecer parado. No contexto atual, *Q-learning* é utilizado para seleção de comportamentos, portanto, estima-se o valor dos pares estado-comportamento $Q(s_t, b_t)$.

Utilizar distribuições de probabilidade na variável de estado do agente implica um espaço de estados contínuo e, portanto, de tamanho infinito. Nessa situação, *Q-learning* deve ser utilizado com métodos de aproximação para que os valores $Q(s_t, b_t)$ possam ser estimados mesmo sendo pequenas as chances de um mesmo estado ser visitado inúmeras vezes. Para esse trabalho, utiliza-se *Q-learning* com aproximação linear de funções, para estimar os valores $Q(s_t, b_t)$, e gradiente descendente, como regra de atualização do vetor de parâmetros utilizando as recompensas obtidas do ambiente.

Para prover capacidade de exploração para o agente, implementa-se um aprendizado ϵ-guloso, isto é, para cada vez que uma ação for selecionada, há uma chance ϵ que a ação obtida não seja ação com maior valor estimado $Q(s_t, b_t)$. Assim, permite-se que o agente aprenda a selecionar comportamentos que antes eram considerados sub-ótimos mas que, então, o permitem alcançar seus objetivos de forma mais eficiente.

3.4 Seleção de ação

O uso de comportamentos permite que, para um mesmo par estado-comportamento, diferentes ações sejam selecionadas. Assim, é necessário definir as regras que mapeiam pares estado-comportamento para ações.

Nesse trabalho, a seleção de ações é realizada de forma algorítmica por meio dos comportamentos de navegação. Assim, define-se um comportamento como uma função capaz de mapear estados para ações. Os pseudocódigos dos comportamentos de fuga, busca e perseguição são apresentados nos Algoritmos 1, 2 e 3, respectivamente.

Algoritmo 1 Comportamento de fuga

```
1: function EXECUTAR_FUGA(s)
2:     acao ← NULL
3:     max_d ← −∞
4:     for all a ∈ Ações do
5:         for all α ∈ Adversarios do
6:             d ← estimar_distancia_apos_acao(a, α)
7:             if d > max_d then
8:                 max_d ← d
9:                 acao ← a
       return acao
```

Algoritmo 2 Comportamento de busca

```
1: function EXECUTAR_BUSCA(s)
2:     acao ← NULL
3:     min_d ← ∞
4:     for all a ∈ Ações do
5:         for all α ∈ Adversarios do
6:             d ← estimar_distancia_apos_acao(a, α)
7:             if d < min_d then
8:                 min_d ← d
9:                 acao ← a
       return acao
```

Algoritmo 3 Comportamento de perseguição

```
1: function EXECUTAR_PERSEGUICAO(s)
2:     acao ← NULL
3:     min_d ← ∞
4:     for all a ∈ Ações do
5:         for all α ∈ Adversarios do
6:             δ ← estimar_deslocamento_adversario(α)
7:             d ← estimar_distancia_apos_acao_com_deslocamento_adversario(a, δ, α)
8:             if d < min_d then
9:                 min_d ← d
10:                acao ← a
       return acao
```

Capítulo 4

Resultados Experimentais

"Não é o conhecimento, mas o ato de aprender; não é a posse, mas o ato de chegar lá, que concede a maior satisfação." Carl Friedrich Gauss

Neste capítulo, descrevem-se os experimentos realizados para validação da solução proposto. Então, apresentam-se os dados obtidos de cada uma das simulações realizadas e a análise desses dados a fim de compreender os resultados dos experimentos.

Os experimentos elaborados nesse trabalho buscam coletar informações que possam trazer luz às seguintes perguntas:

- A metodologia proposta permite que os agentes aprendam a selecionar comportamentos em um sistema multiagente de forma a atingir seus objetivos?
- A metodologia proposta permite que o aprendizado ocorra em um sistema multiagente estocástico?
- A metodologia proposta permite que as políticas aprendidas explorem a cooperação entre os agentes?

4.1 Ambiente de Testes

A validação inicial da metodologia aqui apresentada foi realizada por meio de simulações computacionais utilizando o jogo *Pac-Man*. Isso se deve à possibilidade de executar testes de forma mais rápida do que utilizando robôs físicos, além de permitir que falhas de projeto encontradas ao desenvolver a metodologia sejam resolvidas isoladamente. Dessa forma, erros são identificados e corrigidos mais rapidamente, poupando esforço se comparado ao uso direto de robôs no mundo real para testes iniciais. Por fim, o uso exaustivo de simulações poupa o *hardware* dos robôs de eventuais colisões, diminuindo o custo total do projeto.

4.1.1 Pac-Man

O jogo *Pac-Man* aproxima-se de um problema de caça-predador e foi utilizado para validar a solução proposta. O jogo, criado por Toru Iwatani para os *arcades* japoneses em 1980, gira em torno do personagem *Pac-Man* - cuja forma é semelhante a um círculo amarelo - que está preso em um arena bidimensional e que deve comer todas as pílulas existentes no mapa para pode avançar de nível. Entretanto, seus adversários, representados por fantasmas, podem capturá-lo ao chegar suficientemente perto e fazê-lo perder o jogo. A única defesa do *Pac-Man* reside nas cápsulas distribuídas no mapa. Ao comê-las, os fantasmas tornam-se frágeis e podem ser capturados pelo *Pac-Man*. Assim, cabe ao jogador controlar o *Pac-Man* para cumprir seu objetivo de comer todas as pílulas do mapa evitando os fantasmas ao longo do caminho.

Entre os motivos da imensa popularidade do jogo, encontra-se o sistema de pontuação. Ao capturar pílulas e outros itens do mapa, o jogador recebe pontos que acumulam ao longo das fases. Ao final, o jogo apresenta um ranking com os melhores jogadores que já utilizaram aquela máquina, o que promove intensa competição entre jogadores do mundo inteiro.

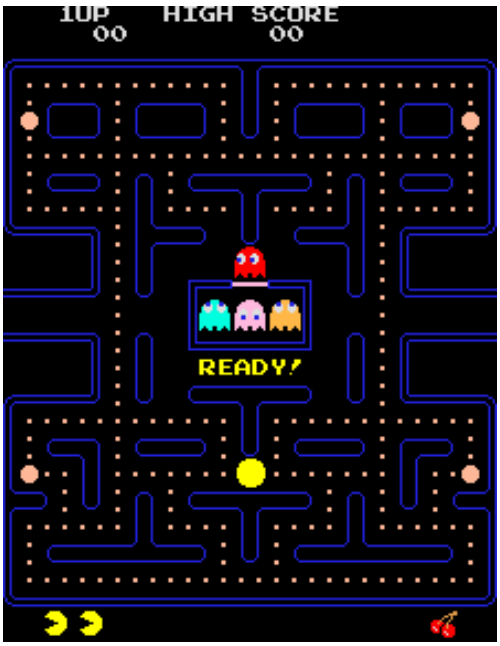


Figura 4.1: Jogo do *Pac-Man* em progresso.

Claramente, o jogo em questão pode ser classificado como um sistema multiagente, uma vez que tanto o *Pac-Man* quanto os fantasmas possuem objetivos específicos e interagem no mesmo ambiente. Sob a perspectiva do *Pac-Man*, o sistema é classificado como um sistema multiagente competitivo, uma vez que a concretização dos objetivos dos fantasmas implicam falha do *Pac-Man*. Já sob a perspectiva dos fantasmas, o sistema é híbrido: ocorre cooperação entre os fantasmas, uma vez que a captura do *Pac-Man* por um fantasma representa a vitória de todos, enquanto ocorre competição contra o *Pac-Man*.

Por possuir pontuação numérica e explícita em todos os momentos, o jogo *Pac-Man* permite fácil integração com algoritmos de aprendizado por reforço. Dessa forma, a função de recompensa,

necessária para que o aprendizado por reforço seja possível, é calculada a partir do histórico de pontos conquistados no decorrer do jogo.

4.1.2 Simulador de Pac-Man

Para simular o jogo *Pac-Man* durante os experimentos, utilizou-se o simulador desenvolvido para disciplinas de IA pelos professores John DeNero e Dan Klein da Universidade de Califórnia, Berkeley. Tal projeto, denominado *The Pac-Man Projects*[1], visa ao ensino de diversas áreas de IA, incluindo aprendizado por reforço. Sua licença de uso permite utilizá-lo e modificá-lo livremente para fins educacionais e não-lucrativos.

Esse simulador – programado em Python 2.7 com suas bibliotecas nativas – permite aos desenvolvedores, de maneira simples e intuitiva, criar agentes inteligentes que percebam o ambiente do jogo e atuem nele. Também é possível criar novos mapas para experimentos e controlar a quantidade de fantasmas que existem no jogo. A Figura 4.2 apresenta o simulador executando um experimento com 3 fantasmas.

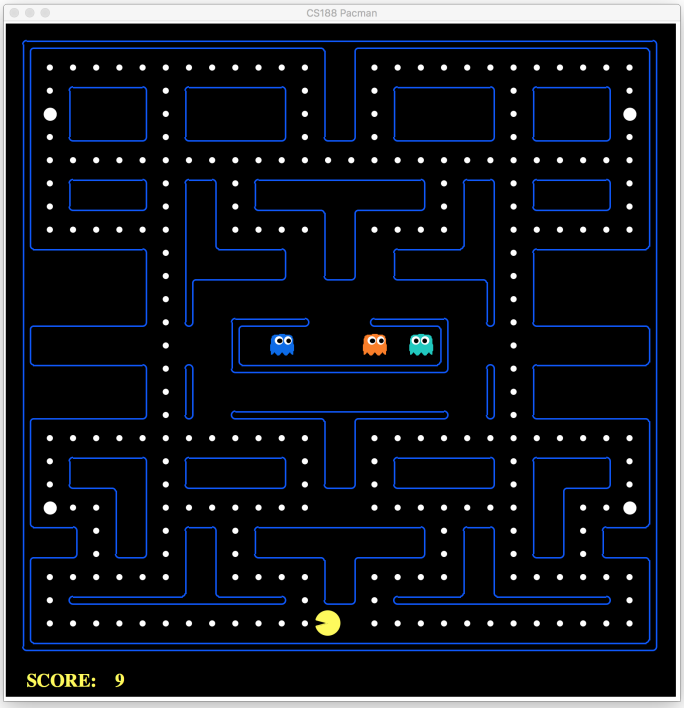


Figura 4.2: Simulador de *Pac-Man* utilizado.

Uma das modificações fundamentais do simulador em relação ao jogo original encontra-se no sistema de pontuação. Alterado para facilitar a implementação algoritmos de aprendizado por reforço, a pontuação pode ser utilizada para calcular as recompensas recebidas pelos agentes. A Tabela 4.1 apresenta como a pontuação varia a cada instante de jogo.

[1] Acessível em: `http://ai.berkeley.edu/project\_overview.html`

Tabela 4.1: Pontuação utilizada no simulador.

Evento	Pontuação
Pac-Man capturado	-500
Penalidade por período	-1
Pac-Man captura pílula	+10
Pac-Man captura fantasma	+200
Pac-Man vence	+500

4.1.3 Arquitetura do sistema

Para realizar os experimentos, desenvolveu-se um sistema[2] que permitisse criar agentes com a metodologia proposta no simulador do jogo *Pac-Man* apresentado. O sistema foi desenvolvido utilizando a linguagem de programação Python na sua versão 2.7. Embora existam versões mais atuais da linguagem, a versão adotada é a que encontra maior uso na comunidade científica e na indústria [39].

O sistema desenvolvido foi projetado para evitar alto acoplamento entre o algoritmo de aprendizado e o simulador. Assim, testes poderiam ser realizados em outros simuladores ou mesmo em robôs no mundo físico com poucas alterações no código escrito. Para realizar tal desacoplamento, o sistema de experimentos foi separado em dois subsistemas independentes conforme apresentado na Figura 4.3: sistema controlador e sistema adaptador.

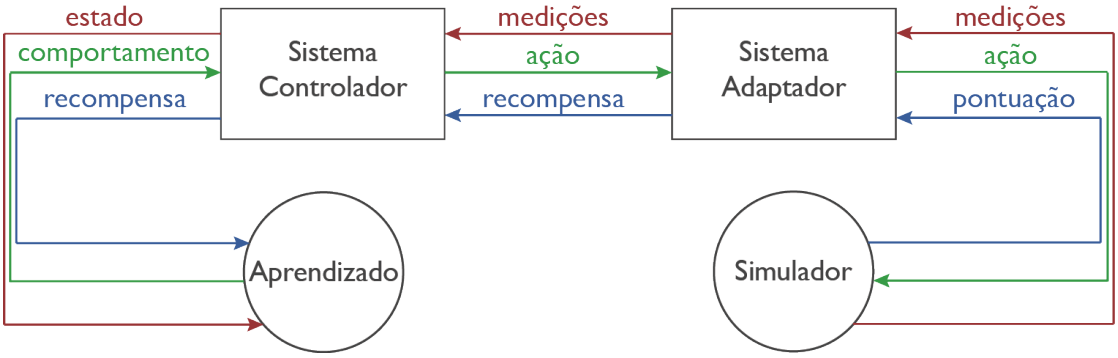


Figura 4.3: Diagrama do sistema desenvolvido.

O sistema controlador recebe as informações de medições e recompensas advindas do adaptador, repassa tais dados para o módulo de aprendizado de cada agente, executa os comportamentos selecionados e envia as ações selecionadas para o sistema adaptador. É importante ressaltar que o módulo de aprendizado é completamente agnóstico à aplicação específica sob a qual os agentes estão submetidos. Dessa forma, é possível utilizar as mesmas rotinas de aprendizado em diferentes cenários sem realizar alterações no seu código. O Algoritmo 4 esboça a rotina implementada no sistema controlador.

[2] Acessível em: `https://github.com/matheusportela/Multiagent-RL`

Algoritmo 4 Rotina do sistema controlador

```
1: while Verdadeiro do
2:     for all agente A do
3:         z ← Leitura dos sensores via Sistema Adaptador
4:         s ← Estimativa do estado utilizando z
5:         ação ← Seleção da ação a partir de s
6:         Enviar ação ao Sistema Adaptador
7:         r ← Recompensa da ação via Sistema Adaptador
8:         Ajuste da seleção de ações utilizando r
```

Por sua vez, o sistema adaptador é responsável por adaptar o ambiente no qual os agentes estão inseridos para respeitar as interfaces disponibilizadas pelo sistema controlador. Assim, o sistema adaptador executa as ações dos agentes, envia as medições do ambiente realizadas pelos sensores dos agentes e calcula a recompensa de cada agente. O Algoritmo 5 apresenta o pseudocódigo da rotina do sistema adaptador.

Algoritmo 5 Rotina do sistema adaptador

```
1: while Verdadeiro do
2:     for all agente A do
3:         z ← Leitura dos sensores
4:         ação ← Resposta do Sistema Controlador para z
5:         Executar ação
6:         r ← Recompensa da ação
7:         Enviar r ao Sistema Controlador
```

Como o sistema adaptador é específico para cada possível aplicação, ele torna-se responsável por executar tarefas que nem sempre são necessárias, tais como armazenar *logs* e interagir com usuários. O sistema controlador, por sua vez, só necessita ser modificado para alterar o algoritmo de aprendizado de comportamentos. Dessa forma, o mesmo sistema controlador pode ser aplicado em inúmeras situações, sejam elas simulações ou experimentos em robôs, bastando modificar o sistema adaptador para atender às necessidades do sistema.

Toda comunicação entre os sistemas é realizada por troca de mensagens utilizando a biblioteca ZeroMQ[3]. Sua escolha se deve à facilidade em implementar diversas arquiteturas de comunicação com poucas modificações, bem como na abstração do protocolo TCP/IP oferecida, permitindo que os sistemas adaptador e controlador possam ser executados, de forma simples, em máquinas diferentes.

4.1.3.1 Agentes simulados

Nos experimentos, todos os fantasmas possuíam algoritmos idênticos de IA, utilizando a abordagem de aprendizado proposta para selecionar os comportamentos de fuga, busca e perseguição.

[3] Acessível em: `http://zeromq.org/`

Por sua vez, o *Pac-Man* foi implementado como um agente aleatório, o qual escolhe suas ações ao acaso.

A quantidade de agentes no ambiente variam conforme o experimento. Apesar de haver sempre 1 agente para o *Pac-Man*, a quantidade de fantasmas variou de 2 a 4 nos experimentos realizados.

4.1.3.2 Variáveis de estado

O estado do sistema é composto pela conjunção de diversas variáveis de estado, cada qual representando a situação atual de um aspecto do ambiente de simulação. As variáveis são:

- Posições dos agentes: Lista de posições $(x \pm \epsilon, y \pm \epsilon)$ dos agentes no ambiente a cada instante de jogo, em coordenadas cartesianas;
- Indicador de fragilidade: Lista de valores lógicos Verdadeiro ou Falso representando se o *Pac-Man* é capaz de capturar os fantasmas;
- Mapa do ambiente: Lista de posições (x, y) das posições que contém pílulas e paredes, representadas em coordenadas cartesianas, recebida apenas no início da simulação.

É importante notar que, matematicamente, a representação em coordenadas cartesianas é equivalente à notação em distância e direção, conforme sugerido em 3.2.1.1. No mapa utilizado para as simulações, $0 \leq x \leq 27$ e $0 \leq y \leq 28$.

A variável ϵ representa o ruído de medição. Nas simulações estocásticas, ϵ é um ruído branco aditivo gerado por meio de uma distribuição uniforme que pode assumir valores entre -3 e 3. Já nos sistemas determinísticos, ϵ permanece constante em 0.

4.1.3.3 Características

O *Q-learning* com aproximação de funções requer que características sejam extraídas a partir do estado estimado do sistema. Portanto, duas características foram implementadas no sistema:

- Distância para cada agente, utilizando grafo de acessibilidade para lidar com obstáculos no ambiente;
- Indicador se *Pac-Man* capturou a cápsula e, portanto, é capaz de capturar o fantasma.

4.1.3.4 Parâmetros de aprendizado

O fator de aprendizado α_t do *Q-learning* com aproximação de funções foi implementado de acordo com a Equação (2.6), com $K = 1$. Mais ainda, definiu-se o fator de desconto como $\gamma = 0.9$, o que faz com que o agente valorize recompensas a longo prazo.

Durante a fase de aprendizado, os fantasmas realizavam exploração ao selecionar comportamentos aleatórios em 10% das vezes. Após finalizado o aprendizado, na fase de testes, os fantasmas deixavam de explorar, selecionando sempre os melhores comportamentos de acordo com as suas estimativas.

4.1.3.5 Recompensas

Utilizando a pontuação $p(k)$, calculada pelo simulador no k-ésimo instante de jogo, a recompensa do agente do *Pac-Man* é calculada pela variação da pontuação, conforme apresentado na Equação (4.1).

$$r_p(k) = \begin{cases} p(k) - p(k-1), \forall k > 0 \\ p(k), k = 0 \end{cases} \tag{4.1}$$

Já a recompensa para os fantasmas, dada a natureza competitiva do sistema em relação ao *Pac-Man*, é calculada pela Equação (4.2), ou seja, é o exato oposto da pontuação do *Pac-Man*. Nota-se que tal pontuação não faz distinção de qual fantasma capturou o *Pac-Man*, caracterizando um sistema multiagente cooperativo entre os fantasmas.

$$r_f(k) = \begin{cases} -(p(k) - p(k-1)), \forall k > 0 \\ -p(k), k = 0 \end{cases} \tag{4.2}$$

4.1.3.6 Dados coletados

Para analisar o desempenho da metodologia proposta, utilizou-se dois indicadores. O primeiro diz respeito à sequência de pontuações ao final de cada jogo de aprendizado e de teste. Assim, os fantasmas estarão alcançando seus objetivos ao minimizar a pontuação do *Pac-Man*.

Mais além, registrou-se também o número de vezes que cada comportamento foi selecionado para cada jogo do experimento. Dessa forma, pode-se avaliar se os fantasmas aprendem comportamentos complementares e exploram a cooperação para alcançar os seus objetivos. Por não haver comunicação entre os agentes, caso ocorra cooperação, ela será do tipo passiva.

4.1.4 Roteiro de experimentos

Buscando responder às perguntas norteadoras, elaborou-se 6 experimentos variando a quantidade de fantasmas no ambiente, bem como a presença de ruído, conforme apresentado na Tabela 4.2.

Para verificar se a metodologia permite que agentes aprendam a selecionar comportamentos em um sistema multiagente, analisou-se as curvas de probabilidade de seleção de comportamentos dos experimentos 1 a 6. Caso as curvas variem ao longo do tempo, há uma forte indicação de que

os agentes estão aprendendo a selecionar comportamentos no sistema. A ausência de aprendizado é indicada por curvas de aprendizado constantes ao longo do tempo.

Os experimentos 4 a 6, os quais contam com a presença de ruído, permitem avaliar se o aprendizado com a metodologia proposta pode ocorrer em um ambiente estocástico. Similarmente, curvas que variam indicam que aprendizado ocorre, enquanto curvas de aprendizado constantes indicam ausência de aprendizado.

A cooperação entre agentes é sugerida, nos experimentos 1 a 6, caso os agentes aprendam a selecionar comportamentos predatórios diferentes. Por exemplo, no experimento 1, um agente aprenderia a utilizar busca enquanto o outro preferiria perseguição.

Em cada experimento, realizou-se 100 jogos durante a fase de aprendizado do algoritmo e 15 jogos durante a fase de testes. Enquanto a fase de aprendizado permite que o agente explore comportamentos considerados sub-ótimos, na fase de testes os agentes devem selecionar os melhores comportamentos de acordo com as estimativas aprendidas ao longo dos jogos.

Tabela 4.2: Descrição dos experimentos realizados.

Número do experimento	**Quantidade de fantasmas**	**Presença de ruído**
1	2	Não
2	3	Não
3	4	Não
4	2	Sim
5	3	Sim
6	4	Sim

4.2 Análise de dados

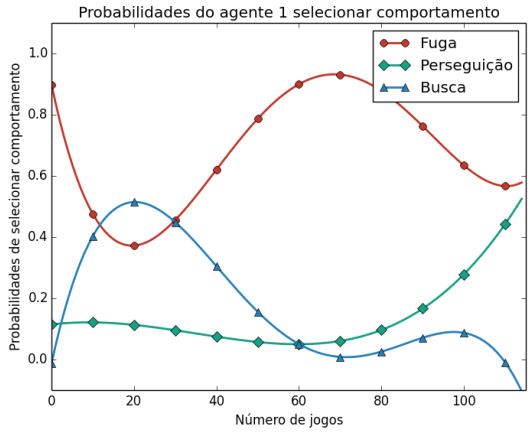
De acordo com os dados coletados, dois tipos de gráficos foram gerados para cada experimento realizado: a probabilidade do agente selecionar cada comportamento disponível, apresentada em conjunto com uma regressão de quarto grau, e as pontuações finais a cada jogo, com uma regressão de primeiro grau.

Pelos testes realizados com o simulador utilizado, a pontuação final do jogo alcança valores próximos de $+3.000$ quando o *Pac-Man* vence a partida. Por outro lado, uma pontuação em torno de -450 indica que o *Pac-Man* perde a partida sem conseguir capturar muitas pílulas.

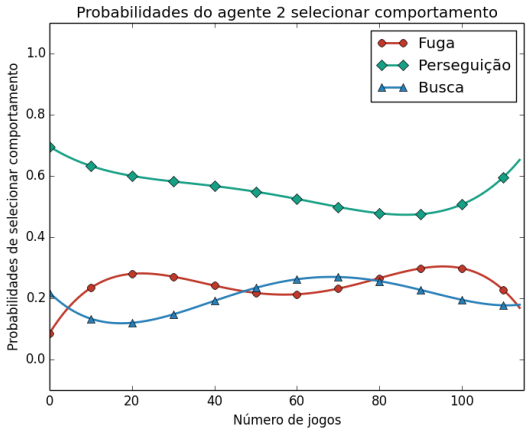
A análise dos resultados é feita em duas etapas. Primeiro, analisa-se os resultados pertencentes aos experimentos realizados em um sistema deterministico, ou seja, sem ruídos das medições. Em seguida, os resultados dos experimentos em um ambiente estocástico serão analisados.

4.2.1 Experimento 1

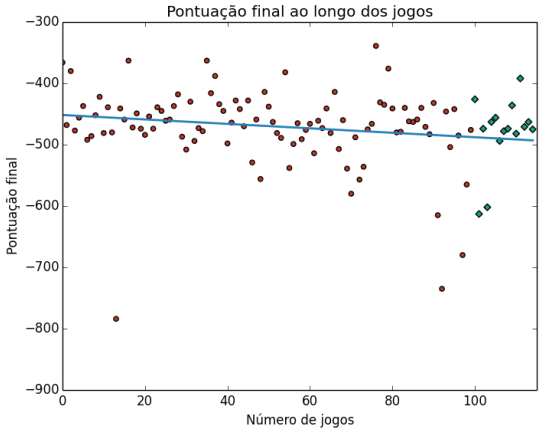
O experimento 1 analisava os dados coletados em um sistema determinístico contendo 2 fantasmas, o qual gerou a curvas de probabilidade de seleção de comportamentos e de pontuação conforme apresentado na Figura 4.4.



(a) Probabilidade de comportamentos do agente 1.



(b) Probabilidade de comportamentos do agente 2.



(c) Pontuação ao longo dos jogos.

Figura 4.4: Resultados do experimento 1.

Primeiramente, é possível notar que as seleções de comportamentos, tanto do agente 1 na Figura 4.4(a), quanto do agente 2 em 4.4(b), variam conforme durante os jogos de aprendizado e de teste, uma vez que as probabilidades de selecionar cada comportamento não mantém-se em torno de um valor constante. Isso significa que as políticas de seleção de comportamentos são alteradas levando em consideração as recompensas recebidas do sistema e que, de fato, os agentes adaptam-se ao ambiente.

Também pode-se notar a diferença entre os comportamentos que são selecionados por cada agente nos jogos de teste, isto é, nos últimos 15 jogos. Enquanto o agente 1 tende a selecionar o comportamento de fuga, o agente 2 claramente prefere escolher o comportamento de busca. Assim, apenas o agente 2 age de forma predatória, visando a captura do *Pac-Man*, enquanto o agente 1

executa ações que não o fazem capturar a sua presa. Portanto, há um indício de que os agentes são capazes de aprender a selecionar comportamentos nesse sistema multiagente, embora não tenham convergidos para comportamentos cooperativos.

Entretanto, sabendo que o comportamento de fuga não leva diretamente à captura do *Pac-Man*, por que o agente o seleciona mesmo assim, em vez de tornar-se mais agressivo e selecionar algum comportamento predatório? Isso pode ser explicado a partir da função de recompensas utilizada: uma vez que ambos os fantasmas recebem recompensa positiva quando o *Pac-Man* é capturado, basta que um agente assuma o papel de predador para que ambos sejam recompensados igualmente. Assim, por mais que o comportamento de fuga não influencie diretamente no objetivo do agente 1, ele recebera reforço positivo e tenderá a selecionar a mesma ação no jogo seguinte.

Mais além, é necessário levar em consideração que os agentes recebem recompensa negativa em duas situações: quando o *Pac-Man* captura um fantasma ou vence o jogo. Em ambas as situações, o comportamento que estiver sendo executado será duramente penalizado e, provavelmente, deixará de ser selecionado no instante seguinte.

Por fim, a seleção do comportamento de fuga também pode ser explicada pelo aprendizado utilizando a abordagem ϵ-gulosa. Durante os jogos de aprendizado, há sempre uma chance ϵ que o agente execute um comportamento aleatório para fins de exploração. Assim, se fosse considerada uma política, o comportamento de fuga seria ocasionalmente escolhido durante a fase de aprendizado. Devido ao caráter probabilístico do aprendizado, é possível que, ao executar o experimento mais vezes, a configuração final da política de seleção de comportamentos convergisse para probabilidades de seleção diferentes.

A pontuação do experimento 1, na Figura 4.4(c), mostra que os jogos finalizam com pontuação em torno de -450, ou seja, a pontuação do *Pac-Man* é severamente reduzida em relação à pontuação máxima obtida durante os testes iniciais no mapa, de cerca de $+3.000$. Embora as pontuações não decaiam de forma significativa através dos jogos, conjectura-se que isso se deve à natureza do *Pac-Man* aleatório.

Sob o ponto de vista dos fantasmas, a situação que gera a maior recompensa é quando o *Pac-Man* anda em uma pequena área sem pílulas por um longo período de tempo, recebendo pontuação negativa a cada passo. Entretanto, como um agente aleatório seleciona ações ao acaso, os fantasmas não são capazes de fazê-lo se comportar dessa forma. Com os comportamentos implementados, os fantasmas preferem então capturar o *Pac-Man* evitando que capture pílulas pelo mapa para aumentar a sua pontuação. Levando essa análise em consideração, os agentes do experimento 1 são aparentemente alcançam o objetivo de minimizar a pontuação do *Pac-Man*.

4.2.2 Experimento 2

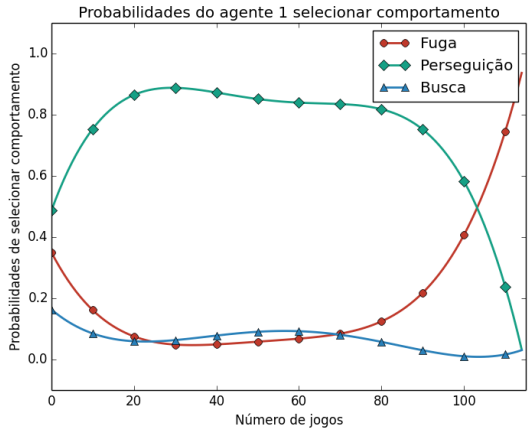
O experimento 2 simulou a situação de um ambiente determinístico com 3 fantasmas. Seus resultados são apresentados na Figura 4.5.

Analisando as políticas de seleção de comportamentos nas Figuras 4.5(a), 4.5(b) e 4.5(c), nota-se a presença de adaptação conforme avançam os jogos simulados. Portanto, o experimento 2

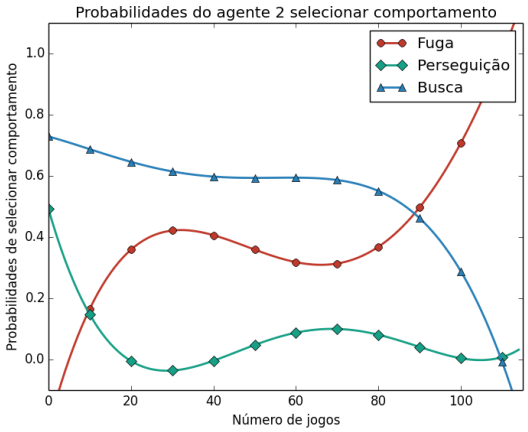
indica que o aprendizado ocorre quando os agentes recebem as recompensas do ambiente, assim como no experimento 1.

É possível notar também que os agentes 1 e 2 adotam uma postura conservadora, preferindo selecionar o comportamento de fugir ao final da aprendizagem. Já o agentes 3 adota postura mais predatória, preferindo o comportamento de busca. Assim como no experimento 1, a presença de pelo menos um agente que selecione comportamentos predatórios, conseguindo capturar o *Pac-Man*, compensa a presença de agentes que não auxiliam nesse processo.

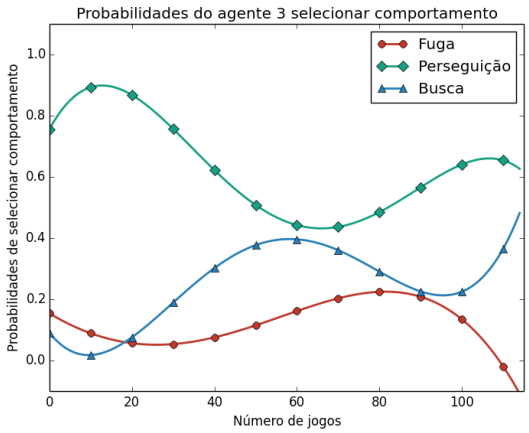
Similar ao experimento 1, as pontuações dos jogos são mantidas constantemente em torno de -450, como apresentado na Figura 4.5(d). Assim, os agentes cumprem os seus objetivos de minimizar a pontuação do *Pac-Man*.



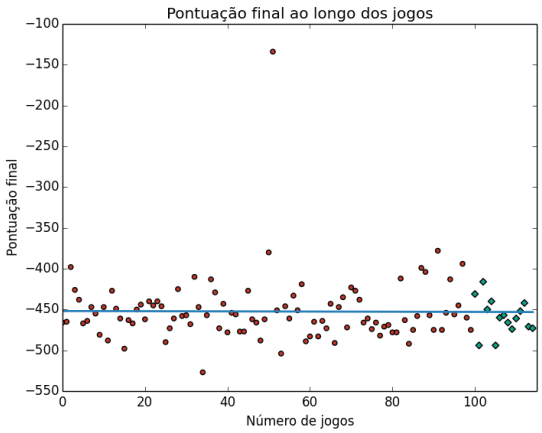
(a) Probabilidade de comportamentos do agente 1.



(b) Probabilidade de comportamentos do agente 2.



(c) Probabilidade de comportamentos do agente 3.

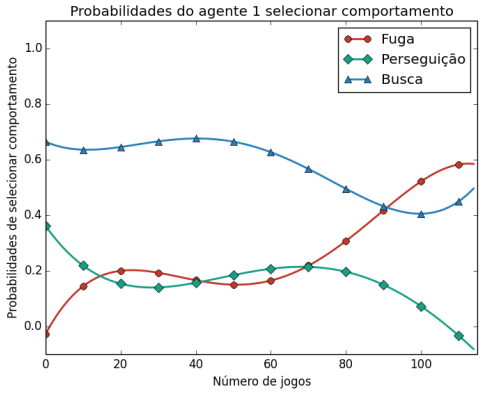


(d) Pontuação ao longo dos jogos.

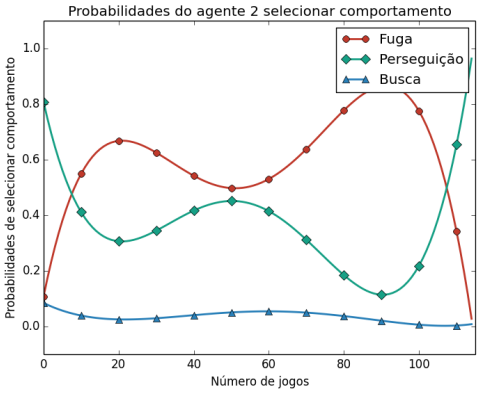
Figura 4.5: Resultados do experimento 2.

4.2.3 Experimento 3

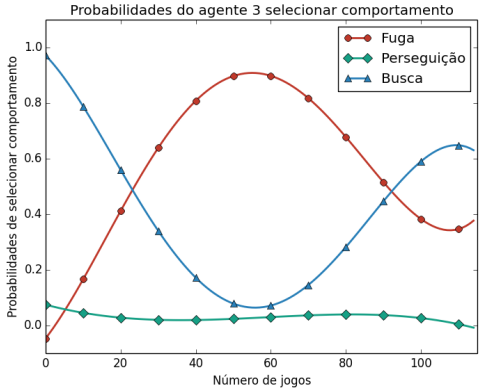
O experimento 3, cujos resultados são apresentados na Figura 4.6, foi realizado com 4 fantasmas em um ambiente determinístico.



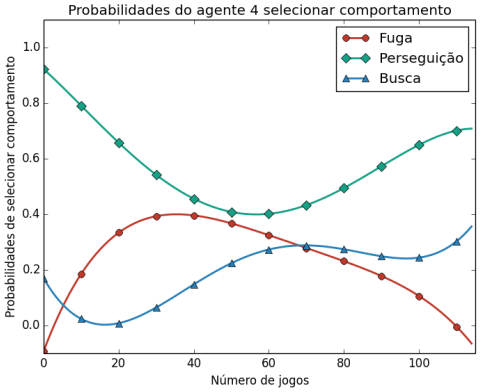
(a) Probabilidade de comportamentos do agente 1.



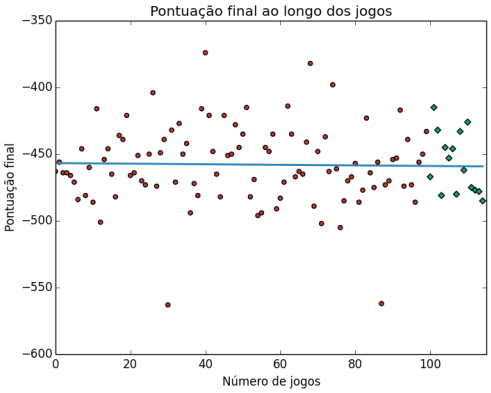
(b) Probabilidade de comportamentos do agente 2.



(c) Probabilidade de comportamentos do agente 3.



(d) Probabilidade de comportamentos do agente 4.



(e) Pontuação ao longo dos jogos.

Figura 4.6: Resultados do experimento 3.

Pela análise das probabilidades de seleção dos comportamentos dos agentes 1, 2, 3 e 4, apresen-

tados nas Figuras 4.6(a), 4.6(b), 4.6(c) e 4.6(d) respectivamente, é possível notar que o aprendizado ocorre pela adaptação da seleção de comportamentos conforme avançam os jogos. Mais ainda, os agentes 2, 3, e 4 possuem maior probabilidade de selecionar comportamentos predatórios, enquanto o agentes 1 prefere o comportamento de fuga ao final do aprendizado. Como discutido anteriormente, a presença de pelo menos um agente com política predadora compensa a ineficiência dos agentes conservadores.

Contudo, é interessante notar que os agentes 2 e 3 convergem para selecionar os comportamentos de perseguição e busca respectivamente. Como tais comportamentos tornam os fantasmas capazes de capturar o *Pac-Man*, cada um de uma maneira diferente, indica-se que os agentes convergiram para políticas cooperativas e complementares. O mesmo ocorre entre os agentes 3 e 4.

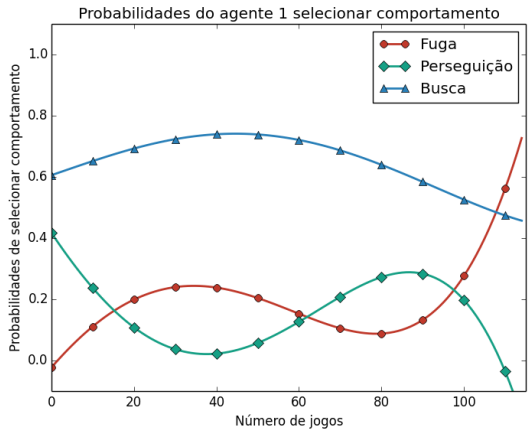
A Figura 4.6(e) demonstra que, mais uma vez, a pontuação final dos jogos fica em torno de -450. Assim, no experimento 4, os fantasmas também são capazes de cumprir o objetivo de minimizar a pontuação do *Pac-Man*.

4.2.4 Experimento 4

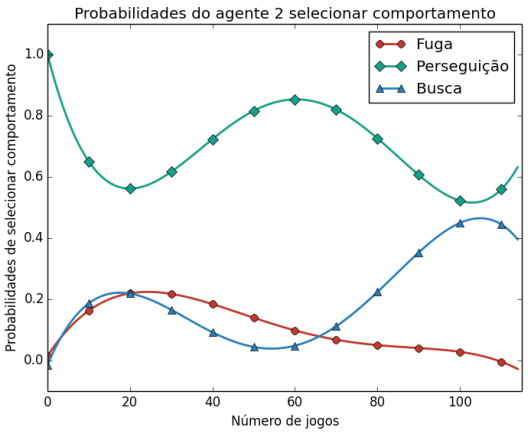
O experimento 4 foi o primeiro a ser realizado em um sistema estocástico, no qual contava com ruídos de medição. Nesse experimento, cujos resultados são apresentados na Figura 4.7, o sistema contava com 2 fantasmas.

Ao analisar as curvas de probabilidade dos agentes 1 e 2 selecionarem cada comportamento, nota-se que as probabilidades de selecionar comportamentos não são constantes ao longo dos jogos, o que indica que o aprendizado por reforço também ocorre em um ambiente estocástico ao receber recompensas do ambiente. Percebe-se ainda que o agente 2 assume um papel predatório, selecionando os comportamentos de busca e perseguição, enquanto o agente 1, embora tenha selecionado o comportamento de busca frequentemente durante o aprendizado, passa a escolher com maior probabilidade o comportamento de fuga ao final da fase de aprendizado. Como apenas um agente utiliza comportamentos capazes de capturar o *Pac-Man*, os dados do experimento 4 não sugerem a existência de cooperação entre os agentes.

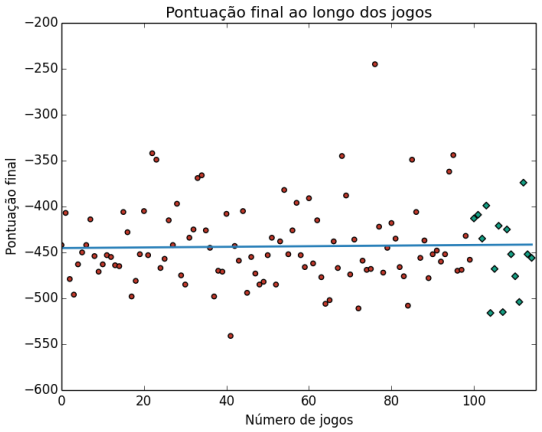
A Figura 4.7(c) demonstra que, mesmo em um ambiente estocástico, os agentes mantém a pontuação do jogo em torno de -450. Possuindo desempenho semelhante ao obtido nos experimentos 1, 2 e 3, os agentes são capazes de cumprir, com sucesso, o objetivo de capturar o *Pac-Man* e minimizar a pontuação do jogo.



(a) Probabilidade de comportamentos do agente 1.



(b) Probabilidade de comportamentos do agente 2.



(c) Pontuação ao longo dos jogos.

Figura 4.7: Resultados do experimento 4.

4.2.5 Experimento 5

O experimento 5 foi realizado simulando 3 fantasmas em um ambiente estocástico. Os resultados dos dados coletados nesse experimento estão na Figura 4.8.

Como nos experimentos anteriores, as probabilidades de seleção de comportamentos variam ao longo dos jogos, conforme apresentado nas Figuras 4.8(a), 4.8(b) e 4.8(c). Portanto, é possível assumir que o algoritmo de aprendizado foi capaz de incorporar as recompensas recebidas e adaptar-se ao ambiente no qual os agentes estavam inseridos.

Esse experimento, destacando-se dos demais, apresenta três agentes com preferência em selecionar comportamentos predatórios. Enquanto o agente 2 especializa-se no comportamento de busca, o agente 3 prefere o comportamento de perseguição. Já o agente 1 possui uma característica mais híbrida, selecionando ambos os comportamentos. Pelos resultados apresentados nesse experimento, os agentes foram capazes de aprender políticas cooperativas e complementares em um ambiente estocástico.

Por fim, a pontuação dos jogos, apresentada na Figura 4.8(d), mais uma vez fica em torno de -450, indicando que os agentes são capazes de minimizar a pontuação do *Pac-Man* em um ambiente estocástico.

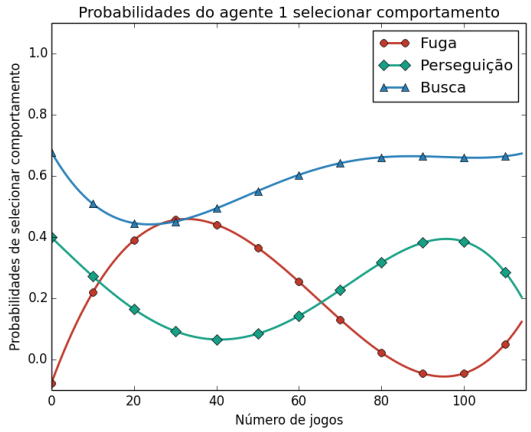
4.2.6 Experimento 6

O último experimento realizado, o experimento 6, contou com 4 agentes inseridos em um ambiente estocástico. Os resultados são apresentados na Figura 4.9.

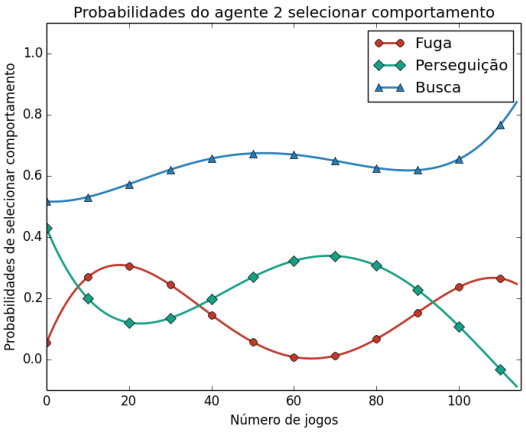
As probabilidades de seleção de comportamentos, apresentadas nas Figuras 4.9(a), 4.9(b), 4.9(c) e 4.9(d), variam ao longo dos jogos, o que indica que o algoritmo de seleção de comportamentos é capaz de incorporar as informações recebidas do ambiente por meio das recompensas.

Nesse experimento, os agentes 1 utiliza principalmente o comportamento de fuga, não auxiliando na captura do *Pac-Man*, já o agente 3, embora também adote o comportamento de fuga, converge ao final selecionando também o comportamento de busca. Embora tais agentes utilizem um comportamento que não auxilia na captura do *Pac-Man*, isso é compensado pelos agentes 2 e 4, que adotam posturas claramente predatórias ao selecionar, principalmente, o comportamento de perseguição e o de busca respectivamente. Assim, o experimento 6 apresenta agentes capazes de aprender políticas cooperativas e complementares em um ambiente estocástico.

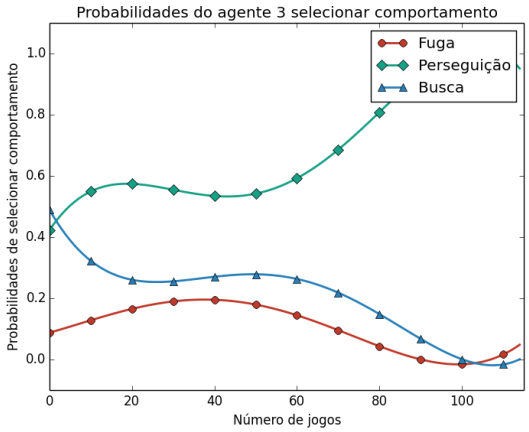
Finalmente, a Figura 4.9(e) mostra que as pontuações final dos jogos ficam em torno de -460, por conseguinte, indicando que os agentes são capazes de alcançar os seus objetivos em um ambiente estocástico.



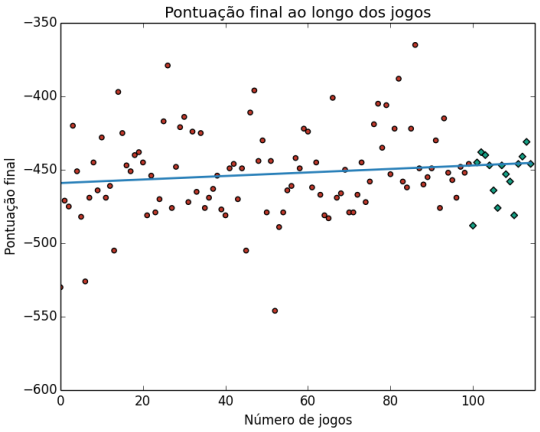
(a) Probabilidade de comportamentos do agente 1.



(b) Probabilidade de comportamentos do agente 2.

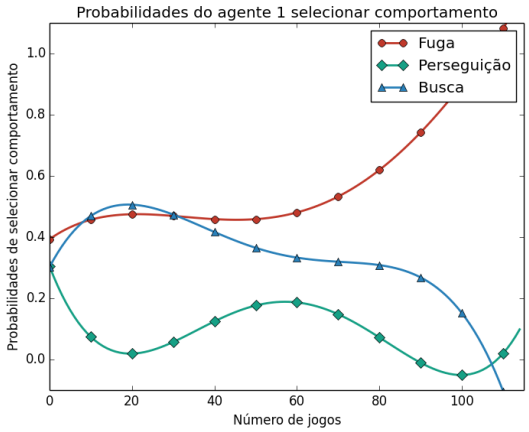


(c) Probabilidade de comportamentos do agente 3.

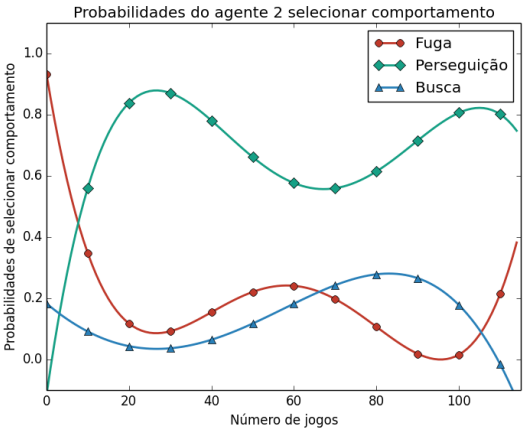


(d) Pontuação ao longo dos jogos.

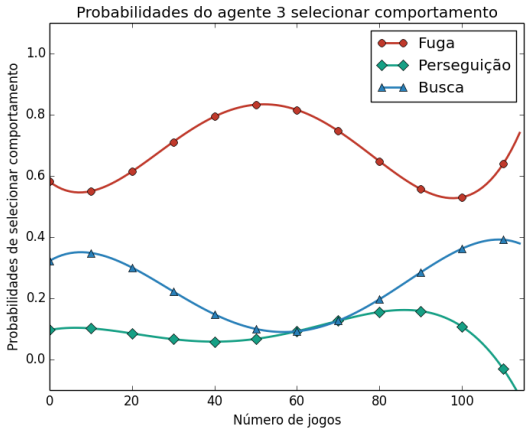
Figura 4.8: Resultados do experimento 5.



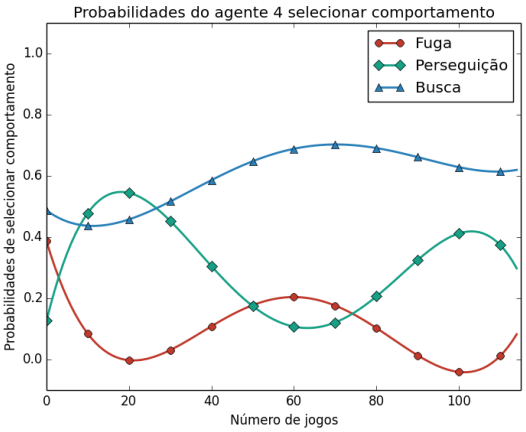
(a) Probabilidade de comportamentos do agente 1.



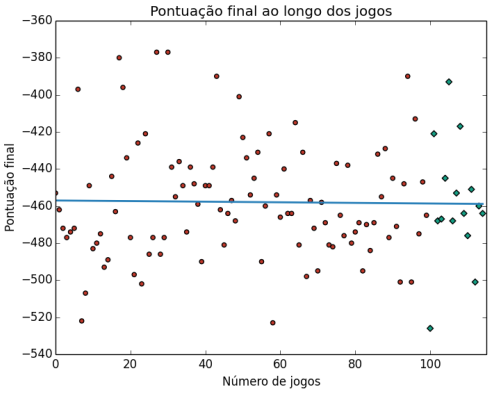
(b) Probabilidade de comportamentos do agente 2.



(c) Probabilidade de comportamentos do agente 3.



(d) Probabilidade de comportamentos do agente 4.



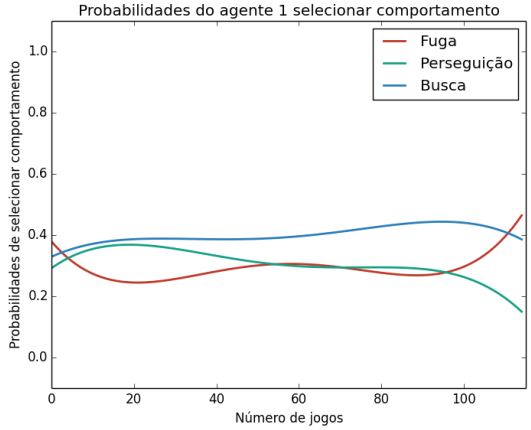
(e) Pontuação ao longo dos jogos.

Figura 4.9: Resultados do experimento 6.

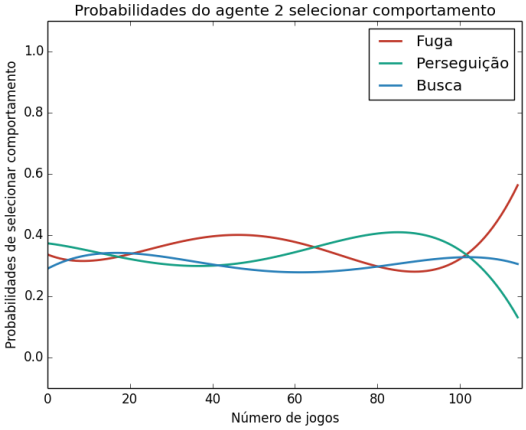
4.2.7 Experimento extra contra *Pac-Man* guloso

Após realizar experimentos variando a quantidade de fantasmas e a estocasticidade do sistema, decidiu-se investigar o desempenho do sistema alterarando o agente do *Pac-Man*. Para tanto, implementou-se um *Pac-Man* guloso, que seleciona ações que o levam em direção às pílulas mais próximas, assim, aumentando a pontuação obtida.

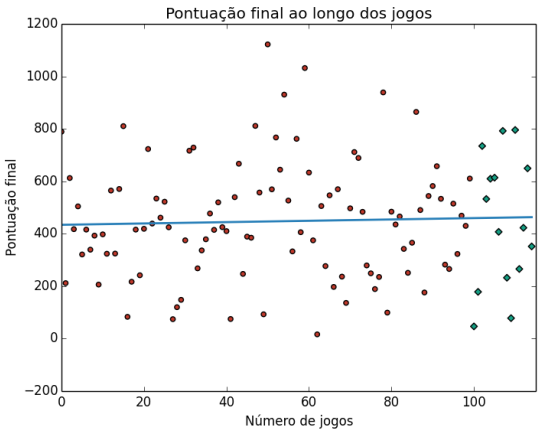
O experimento foi realizado em duas fases, cada uma contendo 2 fantasmas e 10 execuções de 115 experimentos: 100 de aprendizado e 15 de teste. O primeiro experimento foi realizado em um ambiente determinístico enquanto o segundo, estocástico. A pontuação do jogo, uma vez que o *Pac-Man* saía frequentemente vitorioso, foi de +3.000 pontos em média.



(a) Probabilidade de comportamentos do agente 1.



(b) Probabilidade de comportamentos do agente 2.



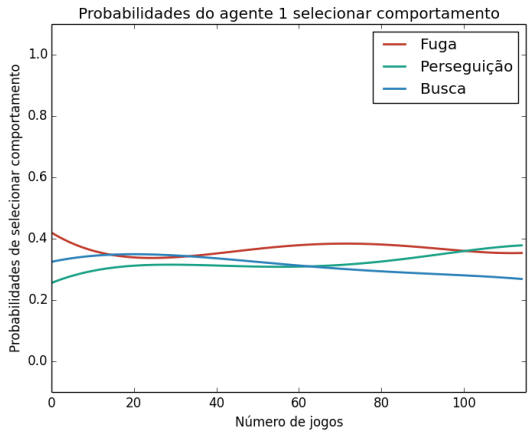
(c) Pontuação ao longo dos jogos.

Figura 4.10: Resultados do experimento extra em ambiente determinístico.

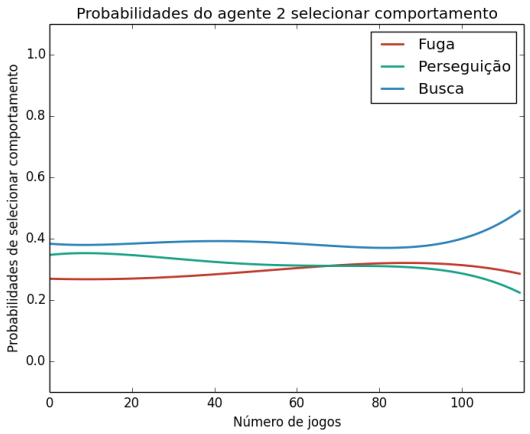
Como é possível notar nas Figuras 4.10(a) e 4.10(b), o agente 1 apresentou um perfil mais predatório do que o agente 2 por possuir maior probabilidade de selecionar o comportamento de busca. Entretanto, de forma inusitada, ambos os agentes possuem altas probabilidades de selecionar o comportamento de fuga. Acredita-se que tal evento tenha ocorrido porque os fantasmas eram capturados com maior frequência nesse experimento do que nos anteriores, valorizando o

comportamento de fuga para se previnir contra esse evento. Devido ao cronograma desse trabalho, foi possível obter apenas resultados preliminares para tais experimentos e, portanto, mais experimentos são necessários para verificar tal hipótese.

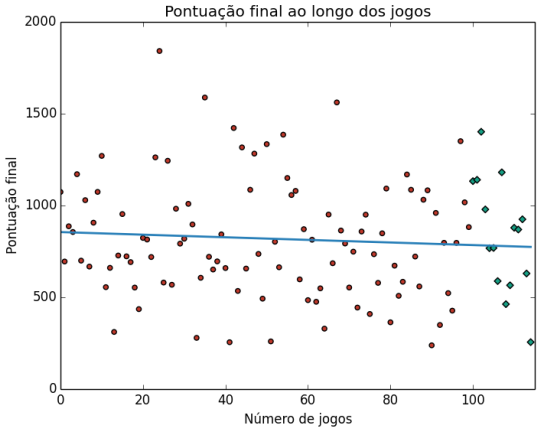
Embora as probabilidades de selecionar os comportamentos possuam um padrão não visto anteriormente, com tendência em selecionar o comportamento de fuga, a Figura 4.10(c) apresenta resultados interessantes. A pontuação do jogo é reduzida de forma significativa, de +3.000 para menos de 500 em média mesmo com ambos os fantasmas selecionando o comportamento de fuga com frequência. Dessa forma, o objetivo de reduzir a pontuação do *Pac-Man* é alcançado contra um agente guloso.



(a) Probabilidade de comportamentos do agente 1.



(b) Probabilidade de comportamentos do agente 2.



(c) Pontuação ao longo dos jogos.

Figura 4.11: Resultados do experimento extra em ambiente estocástico.

Em seguida, avaliou-se o desempenho dos fantasmas contra o *Pac-Man* guloso em um ambiente estocástico, conforme apresentado na Figura 4.11.

As Figuras 4.11(a) e 4.11(b) apresentam o perfil dos agentes ao longo das sessões de aprendizado e teste. É possível notar que o agente 2 age de forma mais predatória, enquanto o agente 1 alterna entre comportamentos predatórios e conservadores. Esse padrão, no qual pelo menos um agente toma a iniciativa predatória, está em conforme com os resultados dos experimentos anteriores.

Já a pontuação dos jogos, conforme é possível analisar na Figura 4.11(c), está em torno de 800 pontos. Mais uma vez, os fantasmas são capazes de cumprir o objetivo de minimizar a pontuação do *Pac-Man*.

4.2.8 Resultados

Os experimentos apresentados foram projetados para coletar dados com os quais pudesse-se responder a três perguntas principais:

- A metodologia proposta permite que os agentes aprendam a selecionar comportamentos em um sistema multiagente de forma a atingir seus objetivos?
- A metodologia proposta permite que o aprendizado ocorra em um sistema multiagente estocástico?
- A metodologia proposta permite que as políticas aprendidas explorem a cooperação entre os agentes?

Por meio da análise realizada nos dados dos experimentos de 1 a 6, as probabilidades de cada agente em selecionar comportamentos foram alteradas ao longo dos jogos. Isso se deve à adaptação do algoritmo de seleção de comportamentos, o qual utiliza recompensas enviadas pelo ambiente para melhorar sua política. Mais ainda, em todos os experimentos, as pontuações dos jogos ficam em torno de -450, indicando que os agentes eram capazes, de capturar o *Pac-Man* e alcançar seus objetivos. Portanto, é possível afirmar que, em todos os experimentos, a metodologia proposta permitiu que agentes inseridos em um contexto multiagente aprendessem a selecionar comportamentos de forma a cumprir os objetivos estabelecidos.

Os experimentos submeteram os agentes tanto a ambientes determinísticos quanto estocásticos, onde havia ruídos de medição. Em ambos os casos, apresentaram capacidade de adaptação e cumpriram seus objetivos da forma esperada. Assim, afirma-se que a metodologia permite que agentes aprendam a selecionar comportamentos em um sistema multiagente estocástico.

Embora os experimentos tenham presenciado agentes que aprendem a selecionar comportamentos diferentes, para fins de análise de cooperação, duas situações ocorreram. Na primeira, parte dos agentes selecionava comportamentos predatórios, como busca e perseguição, enquanto outra escolhia o comportamento de fuga, que em nada auxiliava na captura do *Pac-Man* e, portanto, não influenciava para alcançar o objetivo. Nessa situação, não há indícios que as políticas aprendidas fossem complementares. Tal efeito não influenciou nas pontuações dos jogos porque todos, em todos os experimentos, houve pelo menos um agente com características predatórias capaz de capturar o *Pac-Man*. Já na segunda situação, os agentes aprenderam a utilizar comportamentos predatórios. Como os agentes influenciam diretamente no cumprimento do objetivo, as políticas aprendidas foram cooperativas. Portanto, não é possível garantir que a metodologia apresentada sempre gere políticas cooperativas a partir do aprendizado dos agentes, embora políticas cooperativas sejam ocasionalmente aprendidas.

Capítulo 5

Conclusões

A robótica é uma área de estudo complexa principalmente por lidar com o mundo físico, o qual é inerentemente estocástico. A necessidade de técnicas que tratem as incertezas de sensores, atuadores e representações computacionais do ambiente motiva a aplicação de abordagens probabilísticas, tais como a programação bayesiana, para estimar estados de um sistema.

Tal sistema pode possuir múltiplos agentes percebendo-o e atuando para alcançar seus objetivos. Caso o sistema multiagente seja cooperativo, os agentes combinam seus esforços para alcançar um objetivo comum. Assim, diversos agentes podem cumprir seus objetivos ao mesmo tempo.

Entretanto, a presença de vários agentes em um mesmo ambiente torna-o dinâmico e, portanto, um comportamento que fora considerado ótimo pode tornar-se sub-ótimo posteriormente. Assim, é agentes necessitam de mecanismos de aprendizado para adaptar-se às mudanças ocorridas no sistema.

Este trabalho investigou o desenvolvimento de um algoritmo de aprendizagem por reforço para seleção de comportamentos em sistemas estocásticos com múltiplos agentes autônomos. Embora o foco das aplicações seja em robótica móvel, o algoritmo proposto é geral e pode ser aplicado também em contextos puramente computacionais onde incertezas estejam presentes.

Para tanto, propôs-se uma abordagem baseada em três etapas: estimação do estado do sistema, utilizando programação bayesiana; seleção de comportamento, por meio de *Q-learning* com aproximação de funções; e seleção de ações, a partir dos algoritmos de comportamentos de navegação. Durante o desenvolvimento do trabalho, publicou-se um artigo científico no SBGames 2015 – XIV Simpósio Brasileiro de Jogos e Entretenimento Digital – entitulado "State estimation and reinforcement learning for behavior selection in stochastic multiagent systems", contendo a fundamentação teórica da metodologia proposta. Por motivos de direitos autorais sobre a publicação, o Apêndice I contém somente a primeira página do artigo.

Os experimentos computacionais realizados em simulações do jogo *Pac-Man* demonstram que os agentes são capazes de aprender a selecionar comportamentos em um sistema multiagente e, assim, atingir os objetivos. Mais ainda, o algoritmo pode ser aplicado a ambientes determinísticos ou estocásticos sem diferença perceptível de desempenho. Entretanto, não há garantias que as

políticas aprendidas pelos agentes sejam cooperativas.

5.1 Trabalhos futuros

As políticas aprendidas pelos agentes não são cooperativas em todos os experimentos realizados. Portanto, é interessante analisar se métodos, como comunicação entre agentes ou compartilhamento de políticas [12], são capazes de gerar políticas cooperativas em todas as execuções da solução aqui proposta.

Uma vez validados em simulações computacionais, torna-se possível realizar testes com tal metodologia em sistemas robóticos físicos para validar a sua capacidade de lidar com situações reais. Dessa forma, será possível verificar se o algoritmo é robusto o suficiente para trabalhar com todos os desafios oriundos da robótica, tais como incertezas provindas de dados sensoriais ruidosos, acionadores imprecisos e limitações na representação do estado do mundo. Mantendo-se no âmbito de jogos eletrônicos, pode-se ainda investigar o desempenho do algoritmo em outros jogos além do *Pac-Man*.

Os agentes foram capazes de aprender comportamentos cooperativos de forma passiva, embora tal situação não tenha ocorrido em todos os testes. É possível que, ao inserir métodos de comunicação entre os agentes, os agentes explorem mais frequentemente cooperação e alcancem resultados melhores. Assim, trabalhos futuros que investiguem a cooperação ativa podem expandir o horizonte da abordagem proposta.

Embora tenha-se utilizado comportamentos de navegação para seleção de ações nos agentes, outras técnicas podem ser aplicadas. Por exemplo, trabalhos futuros podem utilizar algoritmos de aprendizado que possibilitem, aos agentes, aprenderem conjuntos de comportamentos de forma autônoma. Assim, retira-se a necessidade do projetista do sistema implementar comportamentos manualmente.

Outro ponto de evolução dos estudos reside na utilização de outros algoritmos, que não *Q-learning* com aproximação de funções, para realizar a seleção de comportamentos. Há a possibilidade de métodos mais complexos como redes neurais obterem resultados melhores.

Trabalhos futuros podem ser realizados para verificar o desempenho do sistema contra diversos outros agentes do *Pac-Man*. Estudos foram iniciados para um agente guloso, embora necessitem de análises mais profundas para compreender alguns resultados inesperados. É possível analisar o desempenho do sistema contra outros agentes mais inteligentes e, até mesmo, em cenário de coevolução. Nessa situação, poderia um agente inteligente do *Pac-Man* melhorar sua performance caso fosse treinado contra os fantasmas inteligentes propostos nesse trabalho?

Por fim, a abordagem proposta não se limita a aplicações robóticas devido ao seu caráter generalista. Assim sendo, trabalhos podem ser desenvolvidos aplicando aprendizagem por reforço em outros cenários estocásticos, tais como controle robusto de plantas industriais, balanceamento de servidores em sistemas distribuídos e análise de risco em papéis financeiros.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] MATARIĆ, M. J. *The robotics primer*. [S.l.]: MIT press, 2007. ISBN 9780262633543.

[2] WIENER, N. *The human use of human beings: cybernetics and society*. [S.l.]: Da Capo Press, 1988. ISBN 9780306803208.

[3] WALTER, W. G. A machine that learns. *Scientific American*, v. 185, n. 2, p. 60–63, 5 1950.

[4] RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Prentice Hall, 2010. ISBN 9780136042594.

[5] THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic robotics*. [S.l.]: MIT press, 2005. ISBN 9780262201629.

[6] ELFES, A. Using occupancy grids for mobile robot perception and navigation. *Computer*, IEEE, v. 22, n. 6, p. 46–57, 6 1989.

[7] KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 1097–1105.

[8] KIM, J.-H. et al. A cooperative multi-agent system and its real time application to robot soccer. In: IEEE. *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. [S.l.], 1997. v. 1, p. 638–643.

[9] LEBELTEL, O. et al. Bayesian robot programming. *Autonomous Robots*, Springer, v. 16, n. 1, p. 49–79, 2004.

[10] WEISS, G. *Multiagent systems: a modern approach to distributed artificial intelligence*. [S.l.]: MIT press, 1999.

[11] MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. *Machine learning: an artificial intelligence approach*. [S.l.]: Springer Berlin Heidelberg, 2013. (Symbolic Computation). ISBN 9783662124055.

[12] PANAIT, L.; LUKE, S. Cooperative multi-agent learning: the state of the art. *Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers, v. 11, n. 3, p. 387–434, 2005.

[13] SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: an introduction*. [S.l.]: MIT press Cambridge, 1998.

[14] BISHOP, C. M. *Pattern recognition and machine learning*. [S.l.]: Springer, 2006.

[15] ALONSO, E. et al. Learning in multi-agent systems. *The Knowledge Engineering Review*, Cambridge University Press, v. 16, n. 03, p. 277–284, 2001.

[16] SKINNER, B. F. *The behavior of organisms: An experimental analysis*. [S.l.]: Appleton-Century, 1938. 457 p.

[17] MARTINSON, E.; STOYTCHEV, A.; ARKIN, R. C. Robot behavioral selection using q-learning. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems2*. [S.l.]: IEEE, 2001. v. 1, p. 970–977. ISBN 0780373987.

[18] REYNOLDS, C. W. Steering behaviors for autonomous characters. In: *Game developers conference*. [S.l.: s.n.], 1999. v. 1999, p. 763–782.

[19] STONE, P.; VELOSO, M. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, Springer, v. 8, n. 3, p. 345–383, 2000.

[20] HAYNES, T.; SEN, S. Evolving behavioral strategies in predators and prey. In: *Adaption and learning in multi-agent systems*. [S.l.]: Springer, 1996. p. 113–126.

[21] GERKEY, B. P.; THRUN, S.; GORDON, G. Visibility-based pursuit-evasion with limited field of view. *The International Journal of Robotics Research*, SAGE Publications, v. 25, n. 4, p. 299–315, 2006.

[22] FLOREANO, D. et al. Design, control, and applications of autonomous mobile robots. In: *Advances in Intelligent Autonomous Systems*. [S.l.]: Springer, 1999. p. 159–186.

[23] NOLFI, S.; FLOREANO, D. Coevolving predator and prey robots: do "arms races" arise in artificial evolution? *Artificial life*, MIT Press, v. 4, n. 4, p. 311–335, 1998.

[24] LITTMAN, M. L. Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the eleventh international conference on machine learning*. [S.l.: s.n.], 1994. v. 157, p. 157–163.

[25] HU, J.; WELLMAN, M. P. Multiagent reinforcement learning: theoretical framework and an algorithm. In: *ICML '98 Proceedings of the fifteenth international conference on machine learning*. [S.l.: s.n.], 1998. v. 98, p. 242–250.

[26] STARANOWICZ, A.; MARIOTTINI, G. L. A survey and comparison of commercial and open-source robotic simulator software. In: ACM. *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments*. [S.l.], 2011. p. 56.

[27] TIKHANOFF, V. et al. An open-source simulator for cognitive robotics research: the prototype of the icub humanoid robot simulator. In: ACM. *Proceedings of the 8th workshop on performance metrics for intelligent systems*. [S.l.], 2008. p. 57–61.

[28] CRAIGHEAD, J. et al. A survey of commercial & open source unmanned vehicle simulators. In: IEEE. *Robotics and Automation, 2007 IEEE International Conference on*. [S.l.], 2007. p. 852–857.

[29] DARWIN, C. *A origem das espécies*. Tradução: John Green. [S.l.]: Martin Claret, 2004. 629 p. ISBN 8572325840.

[30] TINBERGEN, N. On aims and methods of ethology. *Zeitschrift für Tierpsychologie*, v. 20, n. 4, p. 410–433, 1963.

[31] DIAS, R. *Fundamentos de sociologia geral*. [S.l.]: Alínea, 2010. 310 p. ISBN 9788575163801.

[32] MURPHY, R. *Introduction to AI robotics*. [S.l.]: MIT press, 2000.

[33] TESAURO, G. Temporal difference learning and td-gammon. *Communications of the ACM*, v. 38, n. 3, p. 58–68, 1995.

[34] WATKINS, C. J.; DAYAN, P. Q-learning. *Machine learning*, Springer, v. 8, n. 3-4, p. 279–292, 1992.

[35] IRODOVA, M.; SLOAN, R. H. Reinforcement learning and function approximation. In: AAAI. *FLAIRS Conference*. [S.l.], 2005. p. 455–460.

[36] KOIKE, C. M. C. e C. *Bayesian approach to action selection and attention focusing: an application in autonomous robot programming*. Tese (Doutorado) — Institut National Polytechnique de Grenoble, 2005.

[37] HÖLLERER, T. et al. Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. In: CITESEER. *Proc. AIMS*. [S.l.], 2001. v. 1, p. 31–37.

[38] PROAKIS, J. G.; MASOUD, S. *Communication systems engineering*. [S.l.]: Prentice Hall, 2002. ISBN 9780130617934.

[39] PYTHON usage survey 2014. `http://www.randalolson.com/2015/01/30/python-usage-survey-2014/`. Acessado em: 24/11/2015.

APÊNDICES

I. ARTIGO PUBLICADO NO SBGAMES 2015

State estimation and reinforcement learning for behavior selection in stochastic multiagent systems

Matheus Vieira Portela[1]
Faculty of Technology
University of Brasília

Guilherme Novaes Ramos[2]
Department of Computer Science
University of Brasília

Abstract

Intelligent agents can act based on sensor measurements in order to fulfill their goals. In dynamic systems, agents must adapt its behavior selection processes to reflect the changing system state since behaviors that previously were considered the best choice may become sub-optimal. Multiple agents that co-exist in the environment is one example of such a dynamic system. The problem is even greater when stochastic systems are considered, since the states the agents are actually in are unknown. This work proposes a learning algorithm for stochastic multiagent systems, in which Bayesian programming is used for state estimation and Q-learning provides learning capabilities to the agents. An experimental setup using electronic games is described to evaluate the effectiveness of this approach.

Keywords: Artificial Intelligence, Multiagent System, Bayesian Programming, Reinforcement Learning, Q-learning, Predator-pursuit

Author's Contact:

[1] matheus.portela@aluno.unb.br
[2] gnramos@unb.br

1 Introduction

Digital games provide a test bed for experimentation and study of Artificial Intelligence (AI), and there has been a growing interest in applying AI in several problems present in them [Lucas 2008]. One of the more interesting and difficult uses is controlling non-playable characters (NPCs), whose behavior should be interesting, believable, and adapt to the player [Lucas 2009]. Such games also present a well defined environment in which complex situations can be simulated for developing solutions for actual problems [Caldeira et al. 2013].

Uncertainty is inherent to real-world applications, most aspects of the environment are not directly measurable and measurements are often noisy. Therefore, descriptions of a state of the environment require some stochastic approach to handle such uncertainties [Koller and Friedman 2009]. Among current stochastic methods, by using Bayesian inference to reach reasonable conclusions in highly noisy environments, Bayesian methods find successful applications in both academic and industrial problems [Koller and Friedman 2009]. In this scenario, Bayesian programming (BP) provides a generic approach for modeling and decision-taking based solely in Bayesian inference [Lebeltel et al. 2004].

An agent is defined as an entity capable of acting and sensing in an environment in order to maximize a performance measurement [Weiss 1999]. In cooperative multiagent systems (cooperative MAS), multiple agents coexist in the same environment and an agent that improves its performance positively affects the performance of other agents in the same environment. Therefore, multiple agents can reach their goals concurrently [Russell and Norvig 2010].

When dealing with cooperative MAS, which are dynamic environments, an agent is expected to adapt its actions based on previous experiences accumulated by interacting with the environment [Russell and Norvig 2010]. The development of learning machines is a major challenge in contemporary AI research, difficult enough to justify the creation of a whole academic area: machine learning (ML). Within this are reinforcement learning (RL), where agents must learn from interaction which actions yield higher numeric rewards, found extensive usage in MAS [Sutton and Barto 1998].

Typically, agents are controlled by periodically selecting the action that will be executed [Martinson et al. 2001]. A control framework using behaviors, pre-programmed sequences of actions, can also be used so as agents can perform complex tasks. In some cases, using behaviors can even accelerate learning processes [Martinson et al. 2001].

Usually, learning algorithms are validated by evaluating their performance on problems well-known by the scientific community. In cooperative MAS, the predator-pursuit problem is such a standard test. Traditionally, an environment is set up with four predators and one prey where the prey must avoid being captured and predators must capture it [Stone and Veloso 2000]. Eletronic games provide an excellent platform for simulating such conditions, and Pac-Man, one of the most famous games in history [Kent 2001], has its gameplay defined exactly as this type of real-world problem.

This work approaches the problem of creating agents that can learn to select the best behaviors in cooperative MAS where stochasticity is present, such as eletronic games, robotics scenarios, and others. This paper is organized as follows. Section 2 presents previous work with points in common with the work here developed. Section 3 summarizes the Bayesian programming theory, and Section 4 discusses reinforcement learning. Section 5 describes the algorithm proposed in this paper using these. Finally, Section 6 explains the validation experiments that will be conducted to verify the algorithm's performance.

2 Related Work

Probabilistic graphical models have been used as a generic framework to allow automated reasoning in real-world applications. By representing the system with models and considering the most probable cases, computers can reach meaningful conclusions even though not all variables are explicit. This framework is currently being used in medical diagnosis, market analysis, natural language processing, communication systems, among others [Koller and Friedman 2009].

Bayesian programming is a generic framework that incorporates uncertainty in the design of intelligent agents, besides providing efficient computation by using conditional independence assumptions [Koike 2005]. Bayesian programming was used previously in CAD systems, environment mapping, and driving assistance systems [Lebeltel et al. 2004].

Modern computing infrastructures are developed by distributing processing across multiple machines, often in different geographic locations, being connected by networks [Shoham and Leyton-Brown 2008]. State-of-the-art multiagent system algorithms consider computers to be individuals capable of sensing and acting autonomously. This approach, extensively used in cloud computing, creates robust and computationally powerful systems that are able to adapt and process millions of requests per second [Weiss 1999].

Multiagent learning has been studied considering particularities for several scenarios, containing or not elements such as competition, communication, and heterogeneity [Stone and Veloso 2000]. Work has been done to investigate differences in independent and cooperative learning [Tan 1993]. Further studies consider learning in behavior-based systems [Matarić 2001] and usage of game-theory for reinforcement learning in competitive MAS [Littman 1994].

In predator-pursuit research, two techniques are usually em-

II. DESCRIÇÃO DO CONTEÚDO DO CD

O CD contém a seguinte estrutura:

- leiame.pdf: Essa descrição do CD;
- codigo.zip: Código escrito para simulações do sistema compactado em formato ZIP;
- resumo.pdf: Arquivo com o resumo e palavras chaves em formato PDF;
- relatorio.pdf: Arquivo com esse relatório em PDF.