Implementação de Agente Racional Autônomo para Aprendizagem de Comportamentos Colaborativos

Pedro S. Cesarino[1] , Guilherme N. Ramos[2]
[1]Faculdade de Tecnologia, Universidade de Brasília
[2]Departamento de Ciência da Computação, Universidade de Brasília

I. Introdução

Dentre as definições existentes para Inteligência Artificial consideramos a abordagem de agente racional. Um agente é algo que age, mas, mais do que isso, ele percebe o ambiente por meio de sensores e age nele por meio de atuadores, é capaz de criar e alcançar metas além de poder operar de modo autônomo [12]. Por exemplo, um agente humano tem os sentidos como sensores e as pernas, braços e boca como atuadores enquanto um agente robótico pode ter câmeras como sensores e motores como atuadores. Um agente racional é aquele que escolhe realizar ações que são melhores para si dado o que ele sabe sobre o ambiente mesmo que esse conhecimento não esteja certo. O importante é que se o conhecimento estivesse correto o agente escolheria ações para chegar ao seu objetivo considerando que seu conhecimento esteja correto [20].

Um sistema é considerado multiagente se existem pelo menos duas entidades de tal modo que o comportamento de uma é mais bem descrito como a maximização de uma medida de desempenho cujo valor depende de outra entidade [12]. Existem sistemas multiagentes cooperativos, quando os objetivos de cada agente podem ser conquistados simultaneamente [19], por exemplo um time de agentes procurando um objeto; e sistemas multiagentes competitivos que ocorre quando os objetivos dos agentes são conflitantes e eles tem que competir para atingir seus objetivos [19], por exemplo um jogo xadrez. Existem também sistemas híbridos, em que existem cooperação e competição, por exemplo, um jogo de futebol.

Um problema clássico da Inteligência Artificial é o da Caça-Predador em que um time de predadores tem de capturar uma presa, configurando um sistema multiagente híbrido [14], por exemplo *drones* perseguindo fugitivos. Este cenário é interessante para aplicação de algoritmos para multiagentes, pois há várias instâncias diferentes que podem representar muitos cenários multiagente. Os trabalhos a seguir investigaram diferentes nuances desse problema: Nolfi e Floreano discutem sobre a co-evolução e a corrida armada [9]. Haynes e Sen mostram a evolução de estratégias usando algoritmos genéticos [3]. Gerkey, Thrun e Gordon criaram estratégias de busca levando em consideração limitações sensoriais [2].

Projetar programas de agentes é implementar a função do agente, isto é, mapear percepções a ações [12]. Existem agentes reativos simples, que respondem diretamente a percepções, reativos baseados em modelos, que guardam o estado interno para controlar alguns aspectos do mundo, baseados em objetivos que agem para conquistar seus objetivos e baseados em utilidade [12]. Todos esses tipos de agente podem melhorar seus desempenhos por meio de aprendizado.

Ao desenvolver agentes passíveis de movimento, deve-se ter claro quais são possíveis de serem realizados devido a suas limitações físicas para que este não acabe tentando realizar ações impossíveis para ele. Deste modo, um algoritmo de aprendizado pode funcionar mapeando os pares de estado comportamento, que é um conjunto de ações que o agente toma seguindo uma regra de acordo com o estado do ambiente [13]. Assim o algoritmo de aprendizado seleciona qual comportamento é melhor para dado estado, reduzindo o tamanho dos estados e, potencialmente, acelerando o aprendizado [7].

Neste trabalho foram desenvolvidos agentes reativos simples e agentes baseados em objetivos, ambos sem aprendizado, com o objetivo de, usando o *framework* de inteligência artificial (detalhado na seção II-E) desenvolvido em [11], responder a pergunta: dado que o algoritmo desenvolvido no *framework* gera aprendizado para os predadores, como os comportamentos das presas influenciam nesse aprendizado?

A hipótese é que quanto mais efetivo for o desempenho da presa melhores serão os comportamentos aprendidos pelo caçador já que existem diferentes comportamentos que podem ser aprendidos. Para isso, foi definido eficácia como a capacidade da presa de ir atrás do seu objetivo e não ser capturada. O melhor cenário seria que tais comportamentos aprendidos fossem cooperativos e complementares.

II. Fundamentação Teórica

Nesta seção serão apresentados conceitos importantes sobre aprendizado de multiagentes e o *framework* utilizado.

A. Aprendizado de Multiagentes

Sistemas multiagentes são os que apresentam dois ou mais agentes atuando em um mesmo sistema ao mesmo tempo, sendo que estes agentes podem interagir uns com os outros de forma cooperativa ou competitiva [10]. Sendo que, em dado momento, um agente pode não saber tudo sobre o ambiente que o outro saiba devido a restrições, por exemplo, de sensores ou comunicação.

O aprendizado em sistemas multiagentes envolve diversas áreas como a robótica, gestão e recuperação da informação e simulações. É necessário o estudo de como agentes podem aprender comportamentos autônomos e racionais, além da interação entre os diversos agentes, pois é impossível prever todos os possíveis cenários que um agente irá encontrar e qual comportamento seria ótimo em dado momento[1].

Um agente estará aprendendo se ele melhorar seu desempenho em atividades futuras após fazer observações do ambiente [12]. Existem diversas abordagens para lidar com o aprendizado, entre elas: redes neurais, *clustering*, algoritmos genéticos, aprendizado por reforço. Neste último são computadas novas hipóteses assim que novos dados são disponibilizados sendo, esse tipo de aprendizado, interessante para sistemas multiagentes, pois neste caso os agentes precisam atualizar seu conhecimento constantemente [1]. Um algoritmo de aprendizado por reforço muito utilizado é o Q-Learning.

B. Q-Learning com aproximação de funções

O Q-Learning é um algoritmo usado em ambientes em que existe algum tipo de recompensa temporal usada como parâmetro de melhora ou piora para os agentes a cada conjunto de ações feitas dado o estado atual. Dado uma ação tomada em certo estado, o Q-learning estima, a partir da recompensa recebida, um valor para cada par estado-ação [18], isto significa que um agente sabe qual recompensa recebera se realizar aquela ação naquele estado. Porém, em ambientes contínuos seria inviável estimar todos os infinitos pares de estado-ação. Para resolver este problema utiliza-se aproximação de funções [15].

Na aproximação de funções os agentes guardam um vetor de parâmetros, ao invés de guardar todos os possíveis valores estado-ação, o qual é utilizado para calcular a estimativa atual do par estado-ação. Nesta situação o aprendizado ocorre quando os agentes atualizam os valores do vetor utilizando o reforço fornecido pelo ambiente [15].

A maioria das situações reais é tão complexa que é impossível acompanhar todos os aspectos do ambiente, isto é, um agente não consegue determinar com exatidão o próximo estado com apenas o estado atual e a ação realizada pelo agente. Sendo um sistema com essa característica classificado como estocástico o que implica na necessidade de quantificar as incertezas usando probabilidade [16].

No aprendizado por reforço o estado precisa ser uma variável conhecida então é necessário usar métodos para lidar com as incertezas [16]. A programação Bayesiana é utilizada para resolver tal problema sendo sua aplicação especialmente importante na robótica que é inerentemente estocástica.

C. Programação Bayesiana

Os programas bayesianos são compostos por uma descrição e uma pergunta. A descrição é a coleção de informação que se tem de um ambiente e é representado por um conjunto de variáveis discretas que dependem de um conhecimento prévio do ambiente e de dados experimentais[5]. A pergunta é, dada uma descrição, a distribuição de probabilidades que permite calcular a probabilidade de certa variável de interesse utilizando as regras de inferência[5].

D. Robótica

A robótica trata da manipulação e percepção do mundo físico através de mecanismos controlados por computador. Para isto ele utilizando sensores para perceber e atuadores para manipular as coisas, se estes não necessitarem de um controlador humano externo são considerados autônomos [16]. Exemplos de robôs autônomos são carros que dirigem sozinhos [6], linhas de montagem industrial [4], robôs humanoides que realizam tarefas humanas [8], entre outros.

As características apresentadas nas subseções anteriores se encaixam muito bem no contexto da robótica. A maioria das aplicações da robótica são sistemas multiagentes, tendo áreas que o aprendizado é desejável como nos carros autônomos. Sendo que em todos os casos é imprescindível o uso de programação probabilística como a programação bayesiana.

Validações para metodologias nesta área costumam ser feitas por meio de simuladores, mesmo que a simulação não represente toda complexidade do mundo real, pois, desta forma, é possível fazer testes de forma muito mais rápida do que a utilização direta de robôs, além de poupar o hardware dos agentes de eventuais danos [17].

E. Simulador de Pac-Man

O simulador do jogo eletrônico *Pac-Man*, desenvolvido para disciplinas de IA pelos professores John DeNero e Dan Klein da Universidade de Califórnia Berkeley[1], foi utilizado para desenvolver o sistema usado no trabalho de Seleção de comportamentos em múltiplos agentes autônomos com aprendizagem por reforço em ambientes estocásticos [11]. No trabalho citado, comportamento foi definido como um conjunto de ações tomadas por um agente dado um estado do ambiente e será usada neste trabalho também.

Este simulador permite, de forma simples, programar agentes inteligentes e modificar o mapa a fim de fazer diferentes análises. Ele permite também a inserção de ruído nos sensores dos agentes permitindo uma análise estocástica. Tal ruído é um valor r escolhido pelo usuário no simulador, então é gerado um valor aleatório entre -r e +r que é somado nas medições do agente.

O jogo oferece um ambiente multiagente híbrido (do ponto de vista dos fantasmas) e competitivo (do ponto de vista do *Pac-Man*) que se encaixa no problema de caça e predador e possui um sistema de pontuação numérica que torna a utilização de algoritmos de aprendizado por reforço possível. Os pontos a cada instante da simulação podem ser vistos na Tabela I. A pontuação mostrada se refere ao desempenho do *Pac-Man*, portanto, como o algoritmo desenvolvido é usado para o aprendizado dos fantasmas, quanto menor o placar maior a recompensa dos fantasmas.

No caso dos parâmetros usados para este trabalho, descrito em III, a pontuação do jogo fica em torno de -400 quando o *Pac-Man* perde sem capturar muitas pílulas e em torno de +1500 quando ele ganha rapidamente.

[1] http://ai.berkeley.edu/home.html

TABLE I: Pontuação utilizada

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

Dados as cinco ações possíveis para os agentes, Norte, Sul, Leste, Oeste e Parar, ressaltando que os fantasmas não podem realizar a ação Parar, foram desenvolvidos três comportamentos de navegação. Busca*(Seek)* em que o agente escolhe a ação que o faça seguir em linha reta para a posição atual do adversário. Perseguição*(Pursue)* em que o agente almeja se deslocar para uma posição futura de seu adversário levando em conta o descolamento atual dele. Por último Fuga*(Flee)* em que o agente escolhe a ação de forma a ficar o mais longe possível de seus adversários. Os dois primeiros foram considerados comportamentos complementares, pois, no escopo do trabalho, estes dois comportamentos se complementam em uma situação de caça [11]. Em um mesmo jogo o fantasma pode mudar de comportamento quando achar interessante cabendo ao aprendizado ensinar qual deles seguir para cada situação.

F. Especificações do Sistema

O sistema foi desenvolvido de tal forma a evitar o acoplamento entre o algoritmo de aprendizado e o do simulador utilizando dois módulos: adaptador e controlador. Assim o algoritmo desenvolvido para o aprendizado pode ser reutilizado em outros simuladores ou em sistemas robóticos com poucas modificações. O controlador recebe informações do estado e recompensa do adaptador e repassa para o algoritmo de aprendizado o estado e recompensa recebendo de volta um comportamento que por sua vez é enviado ao adaptador que repassa para o simulador e ele devolve ao adaptador uma pontuação e novas medições [11].

III. Proposta e Desenvolvimento

O objetivo do trabalho é investigar como diferentes competidores influenciam no aprendizado dos fantasmas. Para tal foram desenvolvidos *Pac-Man* com comportamentos diferentes. Então, definiu-se o ambiente de testes da seguinte forma: o mapa será um *grid* de 9x18, mapas maiores causariam uma demora maior nas simulações com pouco impacto no aprendizado, com paredes intransponíveis contendo 97 pílulas de comida para a presa, como na Figura 1. Cada jogo acaba apenas quando a presa é capturada ou quando ela come todas as pílulas do mapa. Além disso, existem duas cápsulas especiais no mapa que tornam os fantasmas frágeis permitindo o *Pac-Man* caçar seus predadores por um período de 13 instantes de jogo.

Dado como funciona os algoritmos de aprendizado por reforço, explicado na seção II-B, quanto mais eficiente for o *Pac-Man* menor vai ser a pontuação dos fantasmas resultando em uma maior pressão do ambiente para que os agentes aprendam comportamentos melhores. Espera-se que um ambiente mais competitivo resulte no aprendizado de comportamentos colaborativos e complementares.

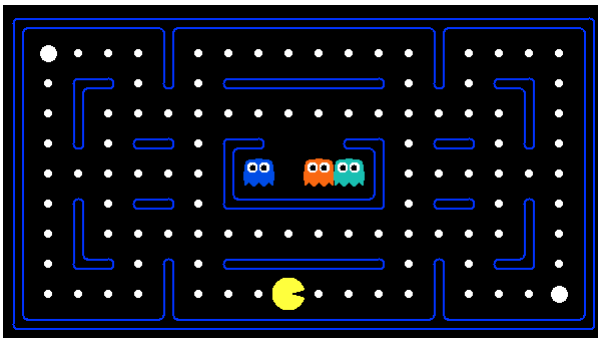

Fig. 1: Mapa do Jogo

A. Pac-Man desenvolvidos

Para criar novos agentes basta definir como se deseja que ele se comporte criando, dentro da classe *Pac-Man* para criar presas ou *Ghost* para criar predadores, a função ''*choose action*", podendo utilizar os comportamentos pré-definidos na seção II-E. Uma explicação detalhada de como desenvolver agentes estão no *GitHub* [2], juntamente com o framework completo.

1) Pac-Man Aleatório: Foi desenvolvido, primeiramente, um*Pac-Man* completamente aleatório com o objetivo de familiarização com o processo de desenvolvimento de agentes na plataforma, porém este agente tinha o problema de ter uma probabilidade muito grande de explorar uma área pequena do total do ambiente, pois não são retiradas das ações sorteadas aquelas que, naquele estado, não gerariam movimento assim facilitando para os fantasmas aprenderem a o capturar facilmente. A imagem 2 mostra um mapa de calor com as posições mais escolhidas pelo agente completamente aleatório em um jogo.

Então outro agente aleatório foi desenvolvido chamado de ''*RandomPacmanAgentTwo*". Este segue as seguintes regras para escolher a sua próxima ação: Escolha uma ação aleatória que resulte em movimento e repita esta ação até chegar a uma parede ou ter mais de duas ações que gerem movimento. Se houverem mais de duas ações o agente pode mudar sua direção ou manter-se no curso anterior. Se houver uma parede repita o primeiro procedimento. O objetivo do desenvolvimento deste foi tentar retirar o problema do completamente aleatório de ficar predominantemente parado já que agora o agente só toma ações que o levam a movimento.

[2]https://github.com/PedroSaman/Multiagent-RL

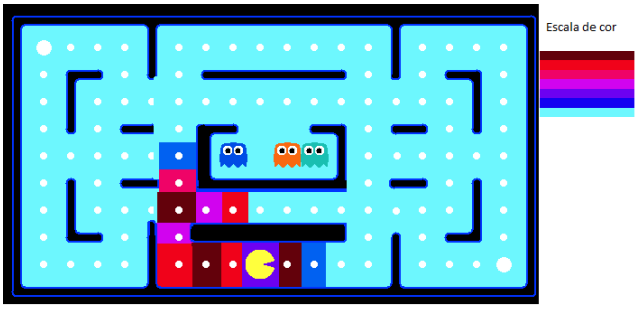

Fig. 2: Mapa de calor

2) Pac-Man Guloso: Para aumentar a eficácia das presas foi proposto um *Pac-Man*, chamado de ''*bfsPacmanAgent*", que fosse atrás do seu objetivo de se alimentar. Ele utiliza a ideia do algoritmo de busca por largura para localizar a comida mais próxima e então, escolhe a ação que o deixa mais próximo dela. O algoritmo foi feito de tal forma a evitar que o agente tome sempre o mesmo caminho nos diferentes jogos simplesmente embaralhando o vetor de ações possíveis.

A busca por largura é uma estratégia de busca em grafo em que o nó mais raso não expandido é escolhido para expansão e para isso usa-se uma fila FIFO (do inglês *First in First out*). Deste modo novos nós vão sempre para o final da fila enquanto nós antigos são expandidos primeiro. Como o algoritmo descarta qualquer caminho novo para um estado já na borda ou dentro do conjunto explorado então tal caminho encontrado deve ser tão profundo quanto ou menor que um já encontrado. Assim este mecanismo de busca resulta no caminho mais raso [12].

A pontuação do *Pac-Man* tende a aumentar já que agora ele vai ativamente atrás do seu objetivo, que é acabar com as pílulas do mapa. Ressaltando que este agente não leva em consideração a existência de adversários então ele irá à direção de seus adversários se a comida mais próxima estiver entre os dois.

3) Pac-Man Ágil: Para resolver o problema do ''*bfsPacmanAgent*", um último agente foi desenvolvido com o intuito de ir atrás de seu objetivo: comer todas as pílulas do mapa, e, ao mesmo tempo, evitar ser capturado. Para isso foi proposto uma algoritmo que levasse em consideração a posição dos fantasmas e a posição da comida mais próxima. Com esse agente, espera-se que sua média de pontuação seja maior que dos agentes anteriores já que agora o *Pac-Man* ativamente evita se capturado.

O algoritmo funciona da seguinte forma: Se nenhum fantasma está a menos de quatro posições, ir atrás da comida mais próxima. Se tiver checar se o fantasma está frágil, se sim ir à direção do fantasma se não tomar a ação que o deixe mais distante possível do fantasma mais próximo. Este foi chamado de ''*NimblePacmanAgent*". Como os fantasmas e o *Pac-Man* tem a mesma velocidade, os fantasmas não capturarão sua presa se a seguirem em linha reta. Portanto, para terem recompensas altas, é esperado que eles buscassem ter comportamentos complementares.

IV. Resultados e Análises

Um experimento foi definido como uma bateria de 30 simulações com os mesmos parâmetros de simulação. Uma simulação corresponde a um conjunto de 115 jogos, dos quais 100 ocorrem o aprendizado por meio do Q-Learning, então terá 15 jogo de teste em que o módulo de aprendizado é desligado e os fantasmas seguem as políticas desenvolvidas, assim como em [11].

Foram feitas simulações com um time formado por um dos *Pac-Man* descritos na seçãoIII-A contra um time de três fantasmas que aprendem. Foram feitos experimentos sem ruído para todos os *Pac-Man* e apenas para o Ágil foi feito, também, um experimento com ruído. Pois, como mostrado em [11], os fantasmas aprendem mesmo em ambientes estocásticos, portanto, como o agente com melhores resultados foi o Ágil, escolheu-se apenas este para analisar se os fantasmas aprendem comportamentos cooperativos mesmo em simulações com um ruído de três.

A. Análise de dados

Foram gerados para cada simulação três tipos de gráfico, um com a probabilidade de cada agente escolher cada um dos três tipos de comportamento de navegação utilizando uma regressão de quarto grau, um com a pontuação final de cada jogo e um com a duração dos jogos em instantes do jogo, ambos com uma regressão de primeiro grau.

Todos os resultados podem ser vistos na tabela II com a pontuação, duração e a quantidade de vezes que cada comportamento foi majoritariamente escolhido em cada simulação. Majoritariedade foi definido como: se um comportamento tem mais de 50% de probabilidade de ser escolhido então ele é o majoritário, se nenhum comportamento for majoritário então ele é indeterminado. Na tabela III, pode-se ver quais foram os dois conjuntos de comportamentos mais escolhidos nos experimentos para cada agente em que B significa busca, P perseguição e F fuga.

TABLE II: Resultados experimentais

Agente	Pontuação	Duração	Busca	Perseguição	Fuga	Indeterminado
Aleatório	-309,19	47,91	32	28	27	3
Guloso	+148,46	54,57	24	33	30	3
Ágil sem ruído	+765,96	117,42	26	23	31	10
Ágil com ruído	-309,48	54,22	22	36	32	0

TABLE III: Conjunto de comportamentos majoritariamente escolhidos

Agente	conjuntos mais escolhidos
Aleatório	BPF, BBP
Guloso	PPF, BFF
Ágil sem ruído	PFF, BBF
Ágil com ruído	PBF, PFF

1) Dados:

2) Análise: Dos agentes desenvolvidos o que obteve o melhor desempenho foi o Ágil, como esperado, já que ele obteve pontuações superiores aos outros além de que seus jogos duraram mais que o dobro para terminar.

Houve menos especialização no experimento com o *Pac-Man* Ágil sem ruído, isto é, os comportamentos aprendidos

pelos fantasmas não convergiam completamente para um único comportamento, mas sim uma combinação dos três disponíveis. Isso aconteceu, pois como a presa é mais eficiente ela chega mais vezes a cápsula especial que a permite capturar seus adversários diminuindo a recompensa deles. Assim eles têm que manter um o comportamento de fuga durante os jogos de teste.

No experimento com o *Pac-Man* ágil com ruído a pontuação abaixou muito, para valores próximos do aleatório, pois o ruído é aplicado nas medições dos fantasmas e do *Pac-Man* e como os fantasmas utilizam da programação bayesiana para tratar das suas informações ele tem como contornar tal problema enquanto o sua presa não, justificando assim esta queda enorme na pontuação.

V. Conclusão

No aprendizado de predadores na situação de caça predador, quanto menos eficientes forem as presas mais fácil se torna o aprendizado. Em situações reais a presa fará de tudo para dificultar sua captura, então para que se tenha um ambiente mais próximo de aplicações reais precisa-se desenvolver, neste contexto, presas que realizem seus objetivos de não serem capturadas.

Desenvolvimento de agentes requer a definição da função do agente mapeando as percepções a ações. Dado um objetivo claro, pode-se desenvolver agentes de diferentes formas a conquistar seu objetivo sendo algumas formas mais eficientes que outras dependendo de um parâmetro. Em uma situação de caça e predador é interessante às presas não serem capturadas então é interessante desenvolver agentes que busquem se manter vivas por maior tempo possível.

Este trabalho propôs o desenvolvimento de agentes para utiliza-los no aprendizado de outros agentes, para investigar como diferentes funções do agente pode impactar o algoritmo de Q-Learning desenvolvido por [11].

Os experimentos computacionais mostraram que diferentemente do que se esperava inicialmente, não houve aprendizado de comportamentos colaborativos e complementares apenas adicionando complexidade e eficácia às presas. Na realidade ocorreram apenas menos especializações e em um ambiente que não se pode prever exatamente o que vai acontecer e com uma mudança tão radical na situação, de repente a presa ser capaz de capturar seu predador, tal especialização poderia ser prejudicial aos predadores.

Outros comportamentos de navegação talvez impliquem em um aprendizado mais interessante e na melhora no desempenho dos fantasmas, além disso, trabalhos futuros podem envolver a utilização de algoritmos que permitam, aos agentes, o aprendizado de comportamentos de forma autônoma. Pode-se, também, fazer análises em cenários de coevolução.

References

[1] Alonso, Eduardo, Mark D'inverno, Daniel Kudenko, Michael Luck e Jason Noble: *Learning in multi-agent systems*. The Knowledge Engineering Review, 16(03):277–284, 2001.

[2] Gerkey, Brian P, Sebastian Thrun e Geoff Gordon: *Visibility-based pursuit-evasion with limited field of view*. The International Journal of Robotics Research, 25(4):299–315, 2006.

[3] Haynes, Thomas e Sandip Sen: *Evolving behavioral strategies in predators and prey*. Adaption and learning in multi-agent systems, páginas 113–126, 1996.

[4] Krüger, Jörg, Terje K Lien e Alexander Verl: *Cooperation of human and machines in assembly lines*. CIRP Annals-Manufacturing Technology, 58(2):628–646, 2009.

[5] Lebeltel, Olivier, Pierre Bessière, Julien Diard e Emmanuel Mazer: *Bayesian robot programming*. Autonomous Robots, 16(1):49–79, 2004.

[6] Levinson, Jesse, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt *et al.*: *Towards fully autonomous driving: Systems and algorithms*. Em *Intelligent Vehicles Symposium (IV), 2011 IEEE*, páginas 163–168. IEEE, 2011.

[7] Martinson, Eric, Alexander Stoytchev e Ronald C Arkin: *Robot behavioral selection using q-learning*. Relatório Técnico, Georgia Institute of Technology, 2001.

[8] Miranda, Cristiana e Yuri Rocha: *Métodos de Comunicação Visual e Controle Cooperativo entre Robôs Humanoides.(2016)*. Trabalho de Graduação Universidade de Brasília. Faculdade de Tecnologia.

[9] Nolfi, Stefano e Dario Floreano: *Coevolving predator and prey robots: Do "arms races" arise in artificial evolution?* Artificial life, 4(4):311–335, 1998.

[10] Panait, Liviu e Sean Luke: *Cooperative multi-agent learning: The state of the art*. Autonomous agents and multi-agent systems, 11(3):387–434, 2005.

[11] Portela, Matheus Vieira: *Seleção de Comportamentos em Múltiplos Agentes Autônomos com Aprendizagem por Reforço em Ambientes Estocásticos.(2015)*. Trabalho de Graduação Universidade de Brasília. Faculdade de Tecnologia.

[12] Russell, Stuart Jonathan, Peter Norvig, John F Canny, Jitendra M Malik e Douglas D Edwards: *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[13] Skinner, Burrhus Frederic: *The behavior of organisms: An experimental analysis*. BF Skinner Foundation, 1990.

[14] Stone, Peter e Manuela Veloso: *Multiagent systems: A survey from a machine learning perspective*. Autonomous Robots, 8(3):345–383, 2000.

[15] Sutton, Richard S e Andrew G Barto: *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[16] Thrun, Sebastian, Wolfram Burgard e Dieter Fox: *Probabilistic robotics*. MIT press, 2005.

[17] Tikhanoff, Vadim, Angelo Cangelosi, Paul Fitzpatrick, Giorgio Metta, Lorenzo Natale e Francesco Nori: *An open-source simulator for cognitive robotics research: the prototype of the iCub humanoid robot simulator*. Em *Proceedings of the 8th workshop on performance metrics for intelligent systems*, páginas 57–61. ACM, 2008.

[18] Watkins, Christopher JCH e Peter Dayan: *Q-learning*. Machine learning, 8(3-4):279–292, 1992.

[19] Weiss, Gerhard: *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[20] Wooldridge, Michael J: *Reasoning about rational agents*. MIT press, 2000.