**UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC**
**CENTRO DE EDUCAÇÃO SUPERIOR DO ALTO VALE DO ITAJAÍ – CEAVI**
**ENGENHARIA DE SOFTWARE**

**ELOÍSA BAZZANELLA**

# EXPANSÃO DA BIBLIOTECA DE APRENDIZAGEM POR REFORÇO PARA DESENVOLVIMENTO DE AGENTES INTELIGENTES PARA A PLATAFORMA NETLOGO

**IBIRAMA**

**2022**

**ELOÍSA BAZZANELLA**

# EXPANSÃO DA BIBLIOTECA DE APRENDIZAGEM POR REFORÇO PARA DESENVOLVIMENTO DE AGENTES INTELIGENTES PARA A PLATAFORMA NETLOGO

Trabalho de conclusão apresentado ao curso de Engenharia de Software do Centro de Educação Superior do Alto Vale do Itajaí (CEAVI), da Universidade do Estado de Santa Catarina (UDESC), como requisito parcial para a obtenção do grau de bacharel em Engenharia de Software.

**Orientador**: Prof. Dr. Fernando dos Santos

**IBIRAMA**

**2022**

## LISTA DE ILUSTRAÇÕES

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | *Application Programmer's Interface* |
| BURLAP | *Brown-UMBC Reinforcement Learning and Planning* |
| CCL | *Center for Connected Learning and Computer-Based Modeling* |
| DT | Diferença Temporal |
| RL | *Reinforcement Learning* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Agentes são sistemas inteligentes, capazes de perceber o ambiente, autônomos em sua tomada de decisão, e aptos a interação com outros agentes. Para a tomada de decisão, o agente pode se basear em deduções lógicas, assim como o raciocínio humano, ou por meio da dedução combinada a algum mecanismo (WOOLDRIDGE, 2009). As simulações baseadas em agentes, por sua vez, fazem uso de agentes simulados para reproduzir e investigar fenômenos. Entre as vantagens de adotar este paradigma de simulação, pode-se mencionar a possibilidade de observar o fenômeno sob duas perspectivas: a do agente, ou seja, diferentes estratégias de tomada de decisão, e a do ambiente, ou seja, os resultados gerados a partir das ações e interações dos agentes (KLÜGL; BAZZAN, 2012).

Para o desenvolvimento de simulações com agentes, existe a possibilidade de implementação da simulação usando linguagens de próposito geral, como por exemplo, o Java. Contudo, atualmente já existem plataformas de simulação que fornecem recursos específicos da área de agentes, que possibilitam simplificar o desenvolvimento, como por exemplo, o NetLogo, o Repast e o MASON. Essas plataformas frequentemente adotam uma linguagem própria de programação. O NetLogo, por exemplo, possui sua própria linguagem e está entre as plataformas mais utilizadas para desenvolvimento de simulações (SKLAR, 2007).

Em simulações baseadas em agentes, os agentes podem incorporar técnicas de inteligência artificial para aprimorarem sua capacidade de se adaptarem às mudanças em si mesmos ou no ambiente. Uma dessas técnicas é a aprendizagem por reforço, que consiste em aprender o que deve ser feito em cada situação, de forma a maximizar a recompensa. Para isto, o agente deve executar diversos testes até encontrar esses valores. Dentro da aprendizagem por reforço, existem diversas técnicas, das quais pode-se mencionar o *Q-Learning* e o *SARSA* (KLÜGL; BAZZAN, 2012). Recentemente, foi disponibilizado uma extensão para a plataforma NetLogo que facilita a incorporação do *Q-Learning* ao comportamento dos agentes (KONS, 2019).

Atualmente, existe uma biblioteca Java chamada *Brown-UMBC Reinforcement Learning and Planning*, também conhecida como BURLAP. Essa biblioteca serve para o desenvolvimento de algoritmos de planejamento e aprendizagem de agentes, disponibilizando algoritmos de aprendizagem por reforço. A BURLAP ainda dispõe de um sistema flexível para a definição dos estados e ações (RAJASINGHAM, 2018).

Contudo, como mencionado, a BURLAP é uma biblioteca disponibilizada para a linguagem Java e o NetLogo, que é a plataforma utilizada para o desenvolvimento de simulações, trabalha com sua própria linguagem. Dessa forma, será refatorada a extensão *Q-Learning* para

que utilize a BURLAP, tendo em vista que a BURLAP abrange vários algoritmos e já está madura e testada.

## 1.1 PROBLEMA

Atualmente, quando surge a necessidade de incorporar técnicas de aprendizagem por reforço em uma simulação na plataforma NetLogo, é possível implementá-la utilizando a atual extensão que disponibiliza mecanismos para incorporar o *Q-Learning* no comportamento do agente. Contudo, quando o desenvolvedor necessita utilizar algum outro método da aprendizagem por reforço, este deverá implementá-lo manualmente.

Um algoritmo como *Q-Learning* é preferível em situações em que o desempenho do agente durante o processo de treinamento não importa, o único objetivo é que o agente aprenda uma política gulosa ideal. Contudo, em situações em que existe uma preocupação referente ao desempenho do agente durante o processo de aprendizagem, é necessário uso de outros algoritmos, como por exemplo, o *SARSA* (JIANG et al., 2019).

Dessa forma, percebeu-se a necessidade de expandir a extensão *Q-Learning* existente de modo a possibilitar aos desenvolvedores, que possuam mais de uma opção de algoritmo de aprendizagem por reforço para incorporar ao comportamento dos agentes em suas simulações.

## 1.2 OBJETIVOS

Esta seção apresenta o objetivo geral e os objetivos específicos do presente trabalho.

### 1.2.1 Objetivo Geral

Expandir a extensão *Q-Learning* para abranger os algoritmos *SARSA* e *Actor-Critic* da aprendizagem por reforço e disponibilizar a extensão atualizada para a plataforma NetLogo, possibilitando aos desenvolvedores de simulações com agentes, incorporar diferentes técnicas de aprendizagem por reforço ao comportamento dos agentes.

### 1.2.2 Objetivos Específicos

a) Definir 3 algoritmos de aprendizagem por reforço que serão disponibilizados.

b) Expandir a extensão Q-Learning para NetLogo incorporando as técnicas de aprendizagem por reforço estudadas, utilizando a biblioteca Java BURLAP.

c) Validar o funcionamento da extensão verificando o aprendizado dos agentes nas simulações.

## 1.3 JUSTIFICATIVA

Implementar agentes que incorporem técnicas de aprendizagem por reforço é uma necessidade enfrentada pelos desenvolvedores de simulações baseadas em agentes. Todavia, codificar este aprendizado pode ser uma grande barreira para os desenvolvedores, sobretudo aqueles que não possuem conhecimentos na área de Inteligência Artificial.

Com o objetivo de facilitar este processo, a extensão existente incorpora o algoritmo Q-Learning ao comportamento dos agentes, disponibilizando uma série de comandos ao desenvolvedor para que configure a aprendizagem. Todavia, a extensão existente abrange um único algoritmo, o que acaba limitando as opções dos desenvolvedores, que ao necessitarem de outro algoritmo, deverão implementá-lo manualmente.

## 1.4 METODOLOGIA

Para atender os objetivos do trabalho, deverá ser realizado um estudo sobre as técnicas de aprendizagem por reforço. O estudo, que será feito a partir de artigos acadêmicos, visa compreender o funcionamento dos algoritmos e a sua popularidade, para então poder selecionar quais algoritmos serão disponibilizados pela extensão. Depois de selecionados, o funcionamento de cada algoritmo deverá ser estudado individualmente.

Também deverá ser estudada a biblioteca Java BURLAP, seu funcionamento e quais artefatos disponibiliza, para que seja possível colocar em prática a utilização da biblioteca durante a produção da extensão.

Com base nos algoritmos que foram selecionados, será realizada a implementação da extensão na linguagem Java utilizando os métodos, previamente estudados, que são disponibilizados pela biblioteca BURLAP.

Após o desenvolvimento, será necessário validar a extensão. Este processo envolve testar a implementação desenvolvida em algum cenário de simulação. Para isso, deverão ser implementadas duas versões de uma mesma simulação para cada algoritmo disponibilizado: uma versão implementando o algoritmo manualmente e outra versão utilizando a extensão, para averiguar se ambas resultam em valores similares de aprendizado.

# 2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os conceitos-chave cujo entendimento é necessário para o desenvolvimento do presente trabalho, tais como as simulações baseadas em agentes, a plataforma NetLogo, a aprendizagem por reforço e alguns de seus algoritmos, a biblioteca Java BURLAP e, por fim, alguns trabalhos correlatos.

## 2.1 SIMULAÇÕES BASEADAS EM AGENTES

Simulações baseadas em agentes é uma abordagem de simulação na qual o sistema sob investigação é representado como um conjunto de agentes. As propriedades mais fundamentais de qualquer agente é a sua autonomia e seu autointeresse. O agente é equipado com um conjunto de metas de alto nível. A cada vez que tem uma escolha de ação, escolhe a ação que acredita ser a que melhor atinge um de seus objetivos. Os agentes também são capazes de perceber e modificar o ambiente em que estão situadas por meio dessas ações (SKLAR, 2007).

Para desenvolver uma simulação baseada em agentes é necessário especificar três elementos: um conjunto de agentes autônomos, as interações entre os agentes e ambiente, e o ambiente que contém todos os demais elementos de simulação (KLÜGL; BAZZAN, 2012). Uma das vantagens em utilizar este paradigma, é a oportunidade de analisar e observar o fenômeno em âmbito microscópico e macroscópico. No âmbito microscópico é possível estudar diversas estratégias de tomada de decisão dos agentes. E em âmbito macroscópico é possível estudar o efeito das ações e interações dos agentes entre si e com o ambiente (KLÜGL, 2008).

A partir da repetida execução da simulação, por um intervalo de tempo predefinido, é possível obter informações significativas a respeito das propriedades do sistema ou fenômeno analisado, bem como sobre sua própria evolução (GARRO; RUSSO, 2010). Por este motivo, a simulação baseada em agentes tem sido usada para modelar e simular sistemas em diversas áreas, tais como agricultura, tráfego, ecologia, economia, assistência à saúde, epidemiologia e redes sociais. Nestes casos, a simulação baseada em agentes foi escolhida como paradigma de simulação pois permite incorporar a complexidade inerente do comportamento dos indivíduos e de suas interações em cenários reais (MACAL; NORTH, 2014).

## 2.2 APRENDIZAGEM POR REFORÇO

Os seres humanos geralmente aprendem por meio de interações com o ambiente. Ao longo da vida, essas interações são uma importante fonte de conhecimento sobre o ambiente e

sobre eles próprios. Aprender por meio da interação é uma ideia fundamental de quase todas as teorias de aprendizagem e inteligência. Quanto à aprendizagem dos agentes, é possível destacar que existem métodos de aprendizado que podem ser incorporados aos agentes para capacitá-los a aprender com o tempo. A aprendizagem por reforço (RL) é um desses métodos, em que o agente aprende através da experiência (SUTTON; BARTO, 2018).

Nas abordagens convencionais de aprendizagem, os sistemas aprendem por meio de pares de entrada e saída, que contribuem com indicativos do comportamento esperado. Neste caso, a tarefa do agente é aprender uma determinada função que consiga gerar estes pares. Porém, para que o agente tenha mais autonomia, ele deverá ser capaz de aprender com base em outras informações, como por exemplo, recompensas ou reforços fornecidos por um "crítico" ou até mesmo pelo ambiente (SERRA et al., 2004).

A RL é uma abordagem da Inteligência Artificial indicada para situações em que se deseja encontrar a política ótima. Uma política é o que especifica ao agente o ele que deve fazer para qualquer estado que possa alcançar, e a política ótima, por sua vez, é a política que traz os melhores resultados. Para alcançá-la, portanto, o indivíduo precisará aprender com base na sua interação com o ambiente, mediante o conhecimento do estado do indivíduo, das ações efetuadas e das mudanças de estado que já ocorreram (SERRA et al., 2004).

Entre as características da RL, é possível mencionar o aprendizado baseado na ação do agente no ambiente que resulta num valor de recompensa, usado para tomar decisões posteriores. Outra característica importante é o *delayed reward*, que diz respeito à qualidade das ações, isto é, um valor de recompensa alto em determinado momento não significa necessariamente que a ação tomada é recomendada, pois o intuito do agente é alcançar objetivos globais e não locais (COSTA; BASTOS, 2012).

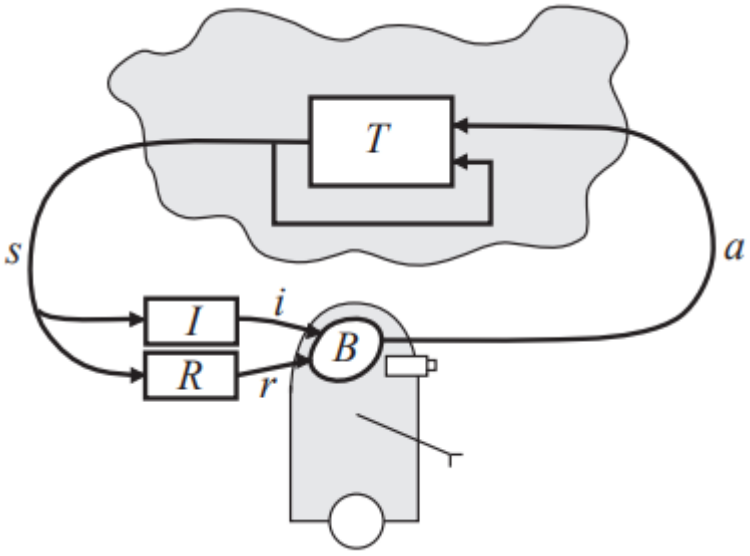
Ademais, a RL também caracteriza-se por ser orientado ao objetivo, ou seja, o agente não precisa conhecer detalhes da modelagem do ambiente, ele simplesmente age neste ambiente desconhecido tentando alcançar um objetivo (SERRA et al., 2004).

Por fim, a última característica da RL é a investigação *versus* exploração, que basicamente consiste em decidir quando se deve, ou não, aprender sobre o ambiente usando a informação já obtida até o momento. A decisão é uma escolha entre agir baseado na melhor informação que existe no momento ou agir para obter novas informações que possibilitem melhores resultados no futuro (COSTA; BASTOS, 2012).

A RL possui um modelo padrão, demonstrado na Figura 1, no qual o agente é conectado ao ambiente por meio da percepção e ação. A cada período da interação, o agente recebe uma entrada $i$ com alguma indicação do estado atual $s$ do ambiente. Com base nisso, o agente escolhe uma ação $a$ para gerar como saída. A ação que foi selecionada pelo agente, altera o estado do ambiente e o valor gerado por essa transição de estado é enviado ao agente por meio

do reforço, $r$. O comportamento $B$ do agente deve selecionar ações que aumentem a soma dos valores de reforço a longo prazo. O agente aprende a fazer isso com o decorrer do tempo através de tentativa e erro, guiados por uma ampla variedade de algoritmos (KRETZSCHMAR; WALLINGA, 2010).

Figura 1 – Modelo padrão de Aprendizagem por Reforço




Fonte: Kaelbling, Littman e Moore (1996).

Para concluir a definição do ambiente, é fundamental especificar uma função de utilidade para o agente. Tendo em vista que o problema é de decisão sequencial, a função de utilidade irá depender de uma sequência de estados. Em cada estado $s$ do conjunto de estados $S$ do problema, o agente receberá uma recompensa $R(s)$ (RUSSEL; NORVIG, 2004). A utilidade do agente é definida em termos da utilidade de sequências de estados. A utilidade de um estado, por sua vez, é a utilidade esperada das sequências de estados que poderiam seguí-lo. Isto é, ela é determinada pela recompensa imediata correspondente ao estado atual, somada à utilidade descontada esperada do próximo estado, considerando-se que o agente escolha a ação ótima. Portanto, a utilidade verdadeira de um estado é denotada por $U(s)$. É de suma importância destacar que $U(s)$ e $R(s)$ são quantidades bastante diferentes; $R(s)$ é a recompensa "a curto prazo" por estar em $s$, enquanto $U(s)$ é a recompensa total "a longo prazo" de $s$ em diante (RUSSEL; NORVIG, 2004).

A aprendizagem por reforço pode ainda ser subdividida em outros métodos, um deles é a aprendizagem de diferença temporal (DT), ou *temporal difference*. Esse método consiste em definir primeiramente as condições que são válidas no estado $s$ quando as estimativas de utilidade estão corretas e, posteriormente, escrever uma equação de atualização que move as estimativas em direção a uma equação de equilíbrio ideal (RUSSEL; NORVIG, 2004).

### 2.2.1 *Q-Learning*

O método de aprendizagem por reforço mais popular é o *Q-learning*, (DAYAN; WATKINS, 1992) que é um algoritmo para estabelecer autonomamente uma política de ações de maneira interativa. É possível demonstrar que o algoritmo *Q-learning* converge para um procedimento de controle ótimo, a partir do momento em que a hipótese de aprendizagem de pares estado-ação Q for demonstrada por uma tabela com as informações do valor de cada par (MONTEIRO; RIBEIRO, 2004).

O *Q-learning* é um método de DT que, ao invés de aprender a utilidade de um estado, aprende uma representação de ação-valor. Utiliza-se $Q(a,s)$ para representar o valor da execução da ação $a$, pertencente ao conjunto ações do problema $A$, no estado $s$. Os valores de $Q$ estão relacionados aos valores de utilidade, onde a utilidade de um estado $s$ é o valor $Q$ máximo entre todas as possíveis ações naquele estado, como demonstrado na função 2.1 (RUSSEL; NORVIG, 2004).

$$U(s) = max\ Q(a,s) \tag{2.1}$$

A função de equilíbrio ideal é uma função de restrição que determina se o agente alcançou uma politica ótima, ou não. Uma política é denotado por $\pi$, e $\pi(s)$ é a ação recomendada pela política $\pi$ no estado $s$. Um política ótima, $\pi^*$, é aquela que produz a utilidade esperada mais alta (RUSSEL; NORVIG, 2004).

O equilíbrio ideal do *Q-learning* é dado pela função 2.2. Nela, $\gamma$ é um fator de desconto que descreve a preferência do agente por recompensas atuais sobre recompensas futuras. $\mathrm{T}(s,a,s')$ é a probabilidade de alcançar o estado $s'$ ao executar a ação $a$ no estado $s$. Por fim, $a'$ é ação com maior valor no estado $s'$.

$$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s')\ max\ Q(a',s') \tag{2.2}$$

A função 2.3, utilizada quando ocorre uma mudança de estado, permite ao agente alcançar o equilíbrio dado pela função 2.1. Nela, $\alpha$ é um parâmetro que representa a taxa de aprendizagem, que define em que medida as informações recém-adquiridas substituem as informações antigas.

$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s) + \gamma\, max_a\ Q(a,s') - Q(a,s)) \tag{2.3}$$

Quanto ao método de seleção da próxima ação, existe uma alternativa que consiste em se comportar de forma gulosa a maior parte do tempo, mas, de vez enquanto, com uma pe-

quena probabilidade $\varepsilon$, selecionar uma ação aleatória do conjunto de ações. Esses métodos de seleção de ação que utilizam essa regra quase gulosa são chamados de $\varepsilon - greedy$ (WUNDER; LITTMAN; BABES, 2010).

### 2.2.2 *SARSA* ($\lambda$)

O algoritmo *SARSA* é uma modificação do algoritmo *Q-Learning* que utiliza um mecanismo de iteração de política, isto é, ele é um algortimo *policy-based*, enquanto o Q-Learning é um algoritmo *value-based* (MONTEIRO; RIBEIRO, 2004). A principal diferença entre os algoritmos *policy-based* e *value-based* é como a regra de atualização se relaciona com a política sendo aprendida pelo agente. (NISSEN, 2007).

O algoritmo SARSA e o Q-learning convergem para diferentes políticas ótimas. O SARSA convergirá para uma solução ótima sob a suposição de que continuará seguindo a mesma política usada para gerar a experiência. Enquanto o Q-Learning convergirá para uma solução ótima sob o pressuposto de que, depois de gerar experiência e treinamento, passará para a política gananciosa (JIANG et al., 2019).

Da mesma forma que no algoritmo *Q-Learning*, o algoritmo *SARSA* também possui uma matriz de transição estado-ação, cuja função de atualização do valores $Q$ está representado pela função 2.4.

$$Q(a,s) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{2.4}$$

O algoritmo *SARSA* converge para uma política e valor de função de ação ótimos quando todos os pares estado-ação tenham sido visitados um número infinito de vezes e a política de escolha da próxima ação tenda a uma política que utilize a melhor ação, isto é, aquela que maximize a recompensa futura esperada (MONTEIRO; RIBEIRO, 2004).

### 2.2.3 *Actor-Critic*

Nos métodos *policy-based*, como o *SARSA*, é construída uma representação de uma política que é mantida na memória durante o aprendizado. Enquanto nos métodos *value-based*, como o *Q-Learning*, não é armazenado nenhuma política explícita, apenas uma função de valor. A política fica implícita e pode ser derivada diretamente da função de valor, ou seja, na escolha da ação com o melhor valor (CALLOWAY, 2019).

Os métodos *value-based* se limitam a um espaço de ações finito. Já os métodos *policy-based*, apesar de possibilitarem um espaço de ações contínuo, precisam ainda executar todo

o episódio para conseguir alguma aprendizagem. Isto faz com que haja uma alta variância e muitas vezes ocasiona lentidão na aprendizagem (CALLOWAY, 2019).

Com o objetivo de superar os problemas que surgiram ao utilizar esses dois métodos individualmente, surgiu o algoritmo *Actor-Critic*, que é basicamente a junção dos dois métodos supracitados. O algoritmo consiste em duas etapas, uma etapa de avaliação de política e uma etapa de melhoria de política. A etapa de avaliação de política diz respeito ao uso eficiente de dados experientes. Enquanto a etapa de melhoria da política serve para melhorar a política em todas as etapas até a convergência, sendo eficiente (PETERS; SCHAAL, 2008). Em outras palavras, no algoritmo *Actor-Critic*, o Crítico realiza a aproximação da função valor, *value-based*, e o Ator realiza a aproximação da função política, *policy-based*, (WANG; CHENG; YI, 2007).

## 2.3 PLATAFORMA NETLOGO

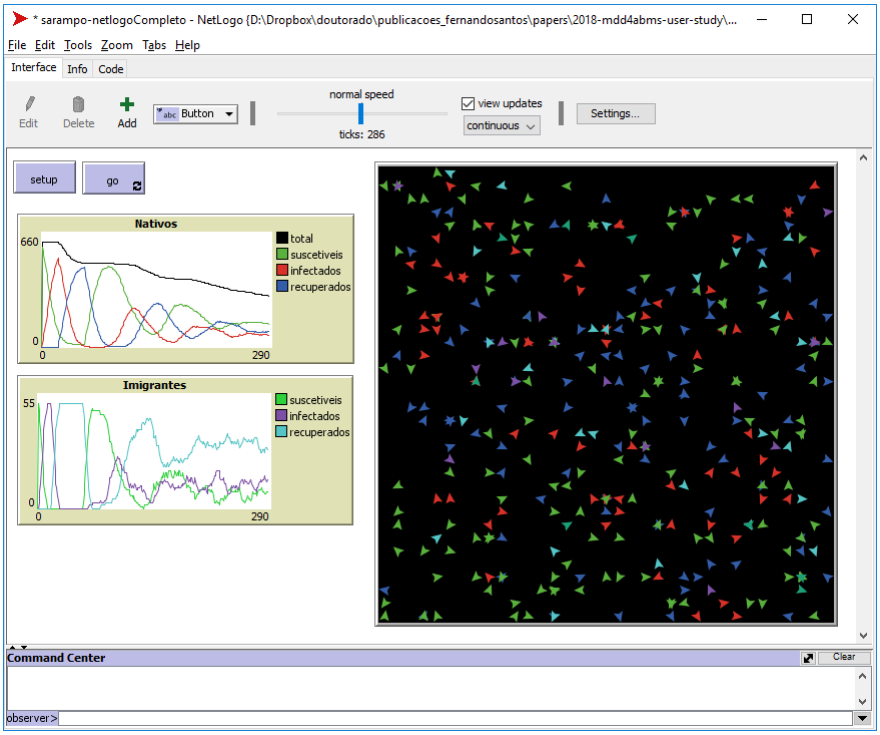
O NetLogo (TISUE; WILENSKY, 2004) é um software gratuito baseado em multiagentes, criado pela *Center for Connected Learning and Computer-Based Modeling* (CCL) da *Northwestern University*. Ele possui sua própria linguagem de programação, com isso, é possível dar instruções para centenas de agentes independentes que agem simultaneamente, com o intuito de explorar as conexões entre os comportamentos de nível micro dos indivíduos e os padrões de nível macro que resultam de suas interações (SKLAR, 2007).

Essa plataforma permite que os usuários possam executar as simulações afim de explorar os comportamentos dos agentes sob diversas condições. O NetLogo é também simples o suficiente para possibilitar que estudantes e pesquisadores criem suas próprias simulações (TISUE; WILENSKY, 2004).

O NetLogo é composto por uma tela bidimensional, como demonstra a Figura 2, chamada de mundo, na qual os agentes, denominados como *turtles*, são apresentados. O desenvolvedor pode alterar o tamanho do ambiente, que é formado por quadrados denominados como *patches*. Isto é, a unidade básica da simulação é a *turtle*, que é o agente. Cada *turtle* reside em um *patch*, ademais, várias *turtles* podem ser localizadas em um único *patch*. Todos os movimentos das *turtles* são calculados em termos de *patches* (SKLAR, 2007).

Além disso, o desenvolvedor pode definir e instanciar sua própria raça de agentes, chamado de *breed*, além de usar o agente genérico de *turtle*. Ainda é possível definir atributos para cada *breed* existente. Outro comando importante a ser mencionado é o *ask*, que é utilizado para dar instruções aos agentes. Um *ask* pode ser feito para todos os agentes, ou para alguma *breed* em específico, ou para algum determinado agente, no último caso deve ser informado junto o identificador do agente desejado (SKLAR, 2007).

Figura 2 – Interface da plataforma NetLogo




Fonte: Santos, Nunes e Bazzan (2018).

Existe uma convenção entre os desenvolvedores dessas simulações que utilizam o NetLogo, que diz respeito à criação de dois procedimentos: um método chamado *setup*, que inicializa a simulação, e um método chamado *go*, que executa cada passo da simulação repetidamente.

A partir da versão 2.0.1 do NetLogo, passou a ser fornecida uma API para a criação de extensões. Desta forma os desenvolvedores podem adicionar novos comandos à linguagem, implementando-os diretamente em Java, Scala ou Kotlin. Isso possibilitou o surgimento de novos tipos de recursos ao NetLogo (TISUE; WILENSKY, 2004).

Recentemente, Kons (2019) desenvolveu e disponibilizou uma extensão para uso do algoritmo *Q-Learning*. O objetivo desta extensão é auxiliar no processo de criação de agentes inteligentes, possibilitando que o desenvolvedor não precise implementar o algoritmo *Q-Learning* manualmente. Contudo, atualmente esta é a única extensão desenvolvida que facilite a criação de agentes inteligentes e restringe-se ao algoritmo *Q-Learning*.

## 2.4 BROWN-UMBC REINFORCEMENT LEARNING AND PLANNING (BURLAP)

A biblioteca de código Java *Brown-UMBC Reinforcement Learning and Planning*, também conhecida como BURLAP, é usada para desenvolver algoritmos RL para sistemas mono e multiagentes. A BURLAP estabelece uma estrutura geral para que os usuários possam definir um domínio de problema. A biblioteca fornece uma ampla variedade de algoritmos RL inte-

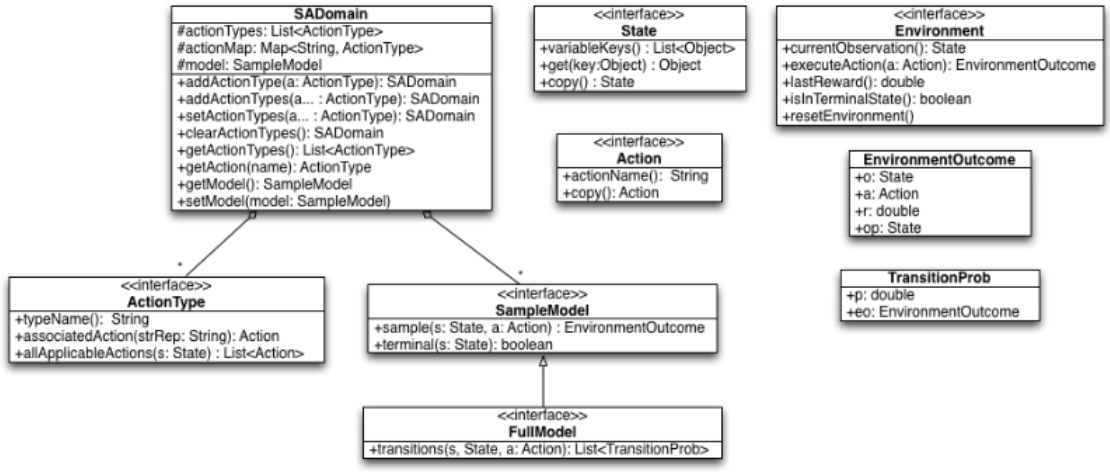
grados, domínios pré-fabricados e ferramentas de visualização (NGUYEN; NGUYEN; NAHAVANDI, 2017).

Entre as vantagens de utilizar a BURLAP, é possível mencionar que ela possui algoritmos que variam desde o planejamento clássico até o planejamento estocástico e algoritmos de aprendizado. A BURLAP também conta com ferramentas para definir visualizações de estados, episódios, funções de valor e políticas. Possui ferramentas para configurar experimentos com vários algoritmos de aprendizado e traçar o desempenho usando várias métricas de desempenho. Possui ainda uma estrutura de *shell* extensível para controlar experimentos em tempo de execução. E por fim, também possui ferramentas para criar torneios multi-agente (MACGLASHAN, 2016).

A BURLAP está disponível no Repositório Central do Maven. O Maven é uma ferramenta de automação de compilação utilizada em projetos Java, ou seja, para adicionar a BURLAP em algum projeto, basta adicioná-la nas dependências. Além disso, no site oficial da biblioteca, existem múltiplos tutoriais que ensinam como utilizá-la (MACGLASHAN, 2016).

Para conseguir utilizar a BURLAP, que pode ser usada com os algoritmos de planejamento ou aprendizagem, primeiramente é necessário se familiarizar com as interfaces Java e estruturas de dados que a biblioteca disponibiliza, apresentadas na Figura 3.

Figura 3 – Diagrama UML das interfaces/classes Java da BURLAP




Fonte: MacGlashan (2016).

A classe *SADomain* é uma estrutura de dados responsável por armazenar as informações sobre o Modelo de Decisão de Markov que será definido e pode ser passada para diferentes algoritmos de planejamento ou aprendizagem. A interface chamada *State*, por sua vez, serve para definir as variáveis de estado, ou seja, uma instância desse objeto especificará um único estado do espaço de estados. Já a interface *Action*, ao ser implementada, serve para definir uma possível ação que o agente pode selecionar.

A interface *ActionType* serve para definir um tipo de fábrica Java para gerar as ações. Esta interface também permite definir pré-condições para as ações, se não houver pré-condições é possível considerar ainda a implementação concreta chamada *UniversalActionType*.

A interface *SampleModel* serve para definir o Modelo de Decisão de Markov. Ela requer apenas que sejam implementados métodos que possam criar uma transição, retornar o próximo estado e recompensar o agente. Já a interface *Environment* serve para fornecer um ambiente com o qual os agentes BURLAP possam interagir. Um outra alternativa é utilizar a classe *SimulatedEnvironment* fornecida, que usa um *SADomain* com um *SampleModel* e simula um ambiente para ele.

A classe *EnvironmentOutcome* contém um estado/observação anterior, uma ação realizada nesse estado, uma recompensa recebida e um próximo estado/observação para o qual o ambiente fez a transição. Por fim, a classe *TransitionProb* contém um objeto do tipo *double* e um *EnvironmentOutcome*, que especifica a probabilidade de ocorrência da transição especificada por *EnvironmentOutcome*.

## 2.5 TRABALHOS CORRELATOS

Esta seção apresenta trabalhos que, de alguma forma, se relacionam com o trabalho que está sendo proposto. No primeiro trabalho desenvolveu-se uma extensão NetLogo para conectar e construir modelos multinível ou multimodelo. No segundo trabalho foi desenvolvido uma biblioteca para controlar o NetLogo através do Python. Por fim, no terceiro trabalho foi desenvolvido uma extensão NetLogo que implementa a arquitetura para base de dados de cognição MetaCiv.

### 2.5.1 *LevelSpace: A NetLogo Extension for Multi-Level Agent-Based Modeling*

No trabalho de Hjorth et al. (2020), foi desenvolvido uma extensão NetLogo que disponibiliza um conjunto de comandos NetLogo para conectar e construir modelos multinível ou multimodelo. Esta modelagem baseada em agente multinível é uma nova abordagem que fornece possibilidades interessantes para a criação de novas formas de modelagem de sistemas conectados. Uma grande característica a ser comentada, é a facilidade de conectar os modelos existentes, pois o LevelSpace funciona abrindo modelos NetLogo de dentro de outros modelos NetLogo sem precisar fazer alterações no código desses modelos. Isso torna mais fácil de expandir ou contestar suposições em modelos existentes.

É demonstrados alguns dos comandos criados e como os comandos se alinham com a linguagem NetLogo existente. Além disso, eles também fornecem exemplos de como usar o

LevelSpace para estruturar processos dentro de modelos existentes ou para conectar fenômenos, possibilitando que sistemas de modelos infinitamente grandes possam se comunicar e interagir entre si.

Por fim, Hjorth et al. (2020) ainda demonstram seis dimensões pelas quais os relacionamentos entre modelos no LevelSpace podem ser descritos e distinguidos de maneira mais detalhada e discutem como essas dimensões ajudam a descrever melhor os modelos NetLogo construídos com LevelSpace.

### 2.5.2 *PyNetLogo: Linking NetLogo with Python*

O trabalho de Jaxa-Rozen e Kwakkel (2018) apresenta a biblioteca pyNetLogo, que serve para controlar o NetLogo através da linguagem de programação Python. Isto é, o pyNetLogo faz a interface do software de modelagem baseado em agente NetLogo com um ambiente Python.

Python é uma linguagem de uso geral e é cada vez mais usado para computação científica pois oferece uma variedade de bibliotecas que podem dar suporte ao desenvolvimento e teste do ABM. Por isso, tendo em vista a popularidade da linguagem Python, o pyNetLogo possui benefícios de um ambiente de análise especializado para um público mais amplo.

Essa funcionalidade é útil quando necessita-se fazer uso de funções estatísticas ou geoespaciais mais avançadas em modelos NetLogo, isto é, a combinação dessas ferramentas permite que os modeladores estendam os recursos do NetLogo com o extenso ecossistema do Python para computação científica.

Por fim, Jaxa-Rozen e Kwakkel (2018) demonstram os recursos utilizando um dos modelos de amostra, isto é, uma amostragem, disponível na ferramenta BehaviorSpace do NetLogo. Para exemplificar as análises mais complexas que são habilitadas por uma interface Python, foi utilizada a biblioteca SALib Python para uma análise de sensibilidade global do modelo. A análise foi realizada usando simulações sequenciais, depois paralelizadas para melhorar o desempenho usando a biblioteca ipyparallel.

### 2.5.3 *CogLogo: une implémentation de MetaCiv pour NetLogo*

O trabalho de Suro, Ferber e Stratulat (2020) se trata de uma extensão NetLogo feita em Java que implementa a arquitetura para base de dados de cognição MetaCiv. O MetaCiv é um framework de Sistemas Multiagentes genérico baseado no meta-modelo MASQ, para a implementação de sistemas sociais complexos.

O CogLogo disponibiliza uma interface gráfica para definir um esquema cognitivo, a relação cognição-plano por meio dos elos de influência. O esquema cognitivo diz respeito ao processo de decisão que leva à escolha do plano a ser executado.

A implementação do plano, por sua vez, é codificada na forma de um procedimento do NetLogo. O CogLogo também fornece um mecanismo de reforço automático, uma escolha de regras para o processo de decisão, peso máximo ou estocástico, bem como a oportunidade de analisar a evolução do comportamento de cada agente.

Por fim, Suro, Ferber e Stratulat (2020) demonstram como realizaram uma simulação utilizando o CogLogo bem como os resultados obtidos.

### 2.5.4 Considerações sobre os Trabalhos Correlatos

O presente trabalho, assim como os trabalhos correlatos mencionados, propõe uma extensão para a plataforma NetLogo. A diferença é que, apesar destes trabalhos correlatos facilitarem de alguma forma o desenvolvimento de agentes, não existe nenhuma relação quanto a aprendizagem por reforço. Nenhum deles se propõe a facilitar a criação de agentes inteligentes, isto porque atualmente existe uma única extensão com este propósito, que é a Q-Learning Extension e esta se limita ao algoritmo Q-Learning. Portanto o presente trabalho se propõe a expandir a Q-Learning Extension de modo a incorporar novos algoritmos.

# 3 EXTENSÃO NETLOGO PARA APRENDIZAGEM POR REFORÇO

As extensões para a plataforma NetLogo têm o objetivo de disponibilizar novas funções por meio de novos comandos para a plataforma, com o intuito de reduzir a complexidade do desenvolvimento. Dessa forma, o presente trabalho propõe-se a expandir a Extensão *Q-Learning*, previamente desenvolvida por Kons (2019), e utilizar a BURLAP para abranger mais algoritmos da aprendizagem por reforço.

A extensão disponibilizará comandos NetLogo para que o desenvolvedor possa especificar o algoritmo que deseja utilizar para o aprendizado, os estados do ambiente, as ações que o agente pode executar, os valores de recompensa com base no estado, um método de seleção da próxima ação e demais valores necessários para a aprendizagem.

As classes que compõe o desenvolvimento da extensão estão representadas no Diagrama de Classes da Figura 4. As classes representadas na cor azul, são as classes disponibilizadas pela BURLAP, que são, em sua maioria, interfaces a serem implementadas. Já as classes representadas na cor vermelha, são as classes disponibilizadas pela API do NetLogo. As classes da extensão, se responsabilizam por criar comandos para serem utilizados no NetLogo, com o intuito de receber as informações vindas das simulações, armazenar essas informações e modelar o problema utilizando a BURLAP.

A classe *ExtensionCommands* determina quais serão os comandos disponibilizados ao NetLogo. Já a classe *LearningRateCommand* é um exemplo de comando a ser disponibilizado, nesta classe é configurado como deve ser a sintaxe para utilizá-lo e o que deve ser feito ao ser executado. As classes *AgentLearning*, *StateDefinition*, *ActionSelection* e *Session*, por sua vez, são responsáveis por armazenar as informações necessárias para a aprendizagem, tais como as possíveis ações, a modelagem dos estados, a função de recompensa, o algoritmo que deve ser utilizado e assim por diante.

Enquanto isso, as classes *Reward*, *IsEndEpisode*, *AgentState* e *AgentStateModel*, usufruem das informações coletadas e armazenadas para realizar a modelagem do problema na BURLAP. Por fim, as classes *QLearningAlgorithm*, *SarsaAlgorithm* e *ActorCriticAlgorithm* são responsáveis por iniciar a criação da aprendizagem e executá-la usando os artefatos disponibilizados pela BURLAP.

Com o intuito de validar o funcionamento da extensão proposta, será implementado duas versões de uma mesma simulação, uma versão implementando o aprendizado manualmente e outra versão utilizando a extensão proposta. Essa validação será realizada para cada algoritmo que será desenvolvido. Quanto à checagem dos resultados da execução, será comparado os resultados de aprendizado de cada versão para averiguar se estão similares.

Figura 4 – Diagrama de Classes


Fonte: Autor (2022).

A simulação escolhida para realizar a validação é o Cliff Walking. Neste cenário o ambiente é um mundo bidimensional, e o tempo é dividido em episódios. O estado do agente é determinado por sua posição atual, isto é, as coordenadas do plano cartesiano. E as ações possíveis que ele pode executar são: ir para cima, ir para a direita, ir para baixo ou ir para a esquerda. O objetivo do agente é partir de um ponto inicial e chegar a um ponto terminal traçando o menor trajeto possível sem cair no penhasco.

Por fim, depois de validada, a extensão será submetida à comunidade de desenvolvedores pelo NetLogo Extension Manager, que é a ferramenta utilizada para encontrar e gerenciar as extensões pelo próprio NetLogo. Para adicioná-la ao NetLogo Extension Manager será necessário adicionar um *Pull Request* ao repositório no GitHub da equipe do NetLogo, que, por sua vez, avaliará a solicitação e tornará a extensão disponível à comunidade.

# 4 CONSIDERAÇÕES FINAIS

O presente trabalho propõe expandir a extensão *Q-Learning* existente, incorporando mais algoritmos e alterando-a para utilizar a biblioteca BURLAP, cujo funcionamento já foi amplamente testado. O intuito da extensão é facilitar o desenvolvimento de agentes inteligentes, disponibilizando funções para que o desenvolvedor possa especificar o algoritmo que deseja incorporar ao comportamento dos agentes, as variáveis necessárias para o funcionamento do algoritmo escolhido, bem como a função de realizar a aprendizagem.

Um dos diferenciais deste trabalho perante os trabalhos correlatos é o seu propósito de facilitar a criação de agentes inteligentes, pois o presente trabalho se propõe a reimplementar o Q-Learning para utilizar a BURLAP, que já é uma biblioteca consolidada e amplamente testada, para realizar a aprendizagem dos agentes de modo a garantir resultados satisfatórios. Outro diferencial é a incorporação de novos algoritmos de aprendizagem por reforço na extensão existente, possibilitando ao desenvolvedor a escolha de qual algoritmo utilizar, ampliando as possibilidades de desenvolvimento.

# REFERÊNCIAS


CALLOWAY, D. **An Introduction to Actor-Critic Deep RL Algorithms**. 2019. Disponível em: <https://www.youtube.com/watch?v=n6K8FfqQ7ds>.

COSTA, G. M.; BASTOS, G. S. Semáforo inteligente–uma aplicação de aprendizagem por reforço. In: **XIX Congresso Brasileiro de Automática**. [S.l.: s.n.], 2012.

DAYAN, P.; WATKINS, C. Q-learning. **Machine learning**, v. 8, n. 3, p. 279–292, 1992.

GARRO, A.; RUSSO, W. easyabms: A domain-expert oriented methodology for agent-based modeling and simulation. **Simulation Modelling Practice and Theory**, v. 18, n. 10, p. 1453–1467, 2010.

HJORTH, A.; HEAD, B.; BRADY, C.; WILENSKY, U. Levelspace: A netlogo extension for multi-level agent-based modeling. **Journal of Artificial Societies and Social Simulation**, JASSS, v. 23, n. 1, 2020.

JAXA-ROZEN, M.; KWAKKEL, J. H. Pynetlogo: Linking netlogo with python. **Jasss**, v. 21, n. 2, 2018.

JIANG, H.; GUI, R.; CHEN, Z.; WU, L.; DANG, J.; ZHOU, J. An improved sarsa ($\lambda$) reinforcement learning algorithm for wireless communication systems. **IEEE Access**, IEEE, v. 7, p. 115418–115427, 2019.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, v. 4, p. 237–285, 1996.

KLÜGL, F. A validation methodology for agent-based simulations. In: **Proceedings of the 2008 ACM symposium on Applied computing**. [S.l.: s.n.], 2008. p. 39–43.

KLÜGL, F.; BAZZAN, A. L. C. Agent-based modeling and simulation. **AI Magazine**, v. 33, n. 3, p. 29–40, 2012.

KONS, K. Biblioteca q-learning para desenvolvimento de simulações com agentes na plataforma netlogo. **Trabalho de Conclusão de Curso. Universidade do Estado de Santa Catarina (UDESC)**, 2019.

KRETZSCHMAR, M.; WALLINGA, J. Mathematical models in infectious disease epidemiology. In: KRÄMER ALEXANDERAND KRETZSCHMAR, M.; KRICKEBERG, K. (Ed.). **Modern Infectious Disease Epidemiology: Concepts, Methods, Mathematical Models, and Public Health**. New York: Springer, 2010. p. 209–221.

MACAL, C.; NORTH, M. Introductory tutorial: Agent-based modeling and simulation. In: **Proceedings of the 2014 Winter Simulation Conference**. Piscataway, NJ, USA: IEEE Press, 2014. (WSC '14), p. 6–20.

MACGLASHAN, J. **Brown-UMBC reinforcement learning and planning (BURLAP)**. 2016.

MONTEIRO, S. T.; RIBEIRO, C. H. Desempenho de algoritmos de aprendizagem por reforço sob condições de ambiguidade sensorial em robótica móvel. **Sba: Controle & Automação Sociedade Brasileira de Automatica**, SciELO Brasil, v. 15, p. 320–338, 2004.

NGUYEN, N. D.; NGUYEN, T.; NAHAVANDI, S. System design perspective for human-level agents using deep reinforcement learning: A survey. **IEEE Access**, IEEE, v. 5, p. 27091–27102, 2017.

NISSEN, S. Large scale reinforcement learning using q-sarsa ($\lambda$) and cascading neural networks. **Unpublished masters thesis, Department of Computer Science, University of Copenhagen, København, Denmark**, 2007.

PETERS, J.; SCHAAL, S. Natural actor-critic. **Neurocomputing**, Elsevier, v. 71, n. 7-9, p. 1180–1190, 2008.

RAJASINGHAM, S. **Implementing efficient planning and learning algorithms for agents in minecraft**. Tese (Doutorado), 2018.

RUSSEL, S. J.; NORVIG, P. **Inteligência Artifical**. 2ª. ed. Rio de Janeiro: Elsevier, 2004.

SANTOS, F.; NUNES, I.; BAZZAN, A. L. Model-driven agent-based simulation development: A modeling language and empirical evaluation in the adaptive traffic signal control domain. **Simulation Modelling Practice and Theory**, Elsevier, v. 83, p. 162–187, 2018.

SERRA, M. R. G. et al. Aplicações de aprendizagem por reforço em controle de tráfego veicular urbano. Florianópolis, SC, 2004.

SKLAR, E. Software review: Netlogo, a multiagent simulation environment. **Journal of Artificial Life**, 2007.

SURO, F.; FERBER, J.; STRATULAT, T. Coglogo: une implémentation de metaciv pour netlogo. 2020.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. 2. ed. [S.l.]: MIT press, 2018.

TISUE, S.; WILENSKY, U. Netlogo: Design and implementation of a multi-agent modeling environment. In: SPRINGER CHAM, SWITZERLAND. **Proceedings of agent**. [S.l.], 2004. v. 2004, p. 7–9.

WANG, X.-S.; CHENG, Y.-H.; YI, J.-Q. A fuzzy actor–critic reinforcement learning network. **Information Sciences**, Elsevier, v. 177, n. 18, p. 3764–3781, 2007.

WOOLDRIDGE, M. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.

WUNDER, M.; LITTMAN, M. L.; BABES, M. Classes of multiagent q-learning dynamics with epsilon-greedy exploration. In: **ICML**. [S.l.: s.n.], 2010.