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 NA PLATAFORMA NETLOGO

IBIRAMA

2022

ELOÍSA BAZZANELLA

EXPANSÃO DA BIBLIOTECA DE APRENDIZAGEM POR REFORÇO PARA DESENVOLVIMENTO DE AGENTES INTELIGENTES NA 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

ELOÍSA BAZZANELLA

EXPANSÃO DA BIBLIOTECA DE APRENDIZAGEM POR REFORÇO PARA DESENVOLVIMENTO DE AGENTES INTELIGENTES NA 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.

Banca Examinadora

Orientador:

Prof. Dr. Fernando dos Santos
UDESC

Membros:

Prof. Dr. Adilson Valdick
UDESC

Prof. Dra. Marília Guterres Ferreira
UDESC

Ibirama, 08/12/2022

Dedico este trabalho à minha irmã, meu bem mais precioso.

AGRADECIMENTOS

Gostaria de demonstrar primeiramente a minha imensa gratidão à minha família, que tanto amo, por todo o apoio e suporte que me deram, não apenas ao longo da faculdade, mas sim ao longo de toda a minha vida! Obrigada por sempre estarem presentes, por me apoiarem em todas as minhas decisões, por me ensinarem o valor da indepedência e, sobretudo, por me ensinarem a voar com minhas próprias asas.

Estendo o meu imenso agradecimento ao meu excelentíssimo namorado, por ser um porto seguro, por sempre estar disposto a me ouvir, por me botar pra cima quando as coisas ficam difíceis e por estar ao meu lado independente das situações. Uma verdadeira graça do acaso encontrar alguém de tão bom coração! Não posso esquecer de agradecer também pelas ajudinhas nos trabalhos e, principalmente, por me aturar em todos os momentos em que estive surtadíssima.

Ademais, sou extremamente grata pela minha parceirinha de faculdade, de ensino médio e de vida! A experiência da graduação foi muito melhor junto a ela, que é uma pessoa tão organizada e comprometida, e que está sempre disposta a fazer as coisas acontecerem. Não existe uma dupla de trabalhos que tenha mais sinergia que nós, digo isso com tranquilidade. Muito obrigada por todos os momentos, amiga! Que possamos continuar crescendo juntas e que possamos viver muitos outros momentos ainda.

Gostaria também de agradecer ao meu querido orientador, por ser sempre tão dedicado, atencioso e paciente. Que foi uma figura fundamental no desenvolvimento desse trabalho e no desenvolvimento do nosso projeto de pesquisa, algo do qual tenho muito orgulho de ter participado, obrigada também por essa oportunidade.

Gratidão aos meus estimados amigos, tanto de dentro, quanto de fora da faculdade. Obrigada por fazerem meus dias melhores, por ouvirem minhas reclamações, e por me distraírem. Vocês são peças essenciais para o aproveitamento da existência!

Por último, mas não menos importante, gostaria de agradecer a mim mesma, por aguentar firme e não desistir em nenhum momento. Por aprender, com o tempo, a não cobrar de si mais do que deveria, por entender que sua métrica de sucesso deve ser a felicidade genuína e por permanecer nessa constante busca do "eu".

"Se você quiser fazer uma torta de maçã do nada, você precisa, primeiro, inventar o universo"
Carl Sagan

RESUMO

Este trabalho apresenta a refatoração e a expansão da extensão *Q-Learning* para a plataforma NetLogo. Essa extensão possibilita aos desenvolvedores de simulações baseadas em agentes, incorporarem técnicas de aprendizagem por reforço ao comportamento dos agentes, de forma simplificada através de novos comandos. Primeiramente foi realizada a refatoração da antiga extensão *Q-Learning* para utilizar a BURLAP, que é uma biblioteca Java que disponibiliza diversos algoritmos de aprendizagem por reforço. Posteriormente, foi realizada a expansão da extensão, incorporando-se os algoritmos SARSA (λ) e *Actor-Critic*. Para validar o funcionamento da extensão, foram feitos experimentos utilizando o cenário *Cliff Walking*, que é um problema clássico da aprendizagem por reforço. Com base nas execuções das simulações, foram coletados os resultados das políticas ótimas aprendidas pelos agentes. Verificou-se que as políticas aprendidas permaneceram consistentes, evidenciando a ausência de falhas na refatoração. Além disso, os novos algoritmos incorporados apresentaram resultados satisfatórios das políticas ótimas aprendidas, conforme o esperado. A extensão desenvolvida está disponível online para uso da comunidade, juntamente com sua respectiva documentação e exemplo de utilização.

Palavras-chave: Simulações Baseadas em Agentes. Aprendizagem por Reforço. Agentes Inteligentes. BURLAP. NetLogo.

ABSTRACT

This paper presents the refactoring and expansion of the Q-Learning Extension for the NetLogo platform. This extension allows agents-based simulation developers to incorporate reinforcement learning techniques into agents' behavior in a simplified way through new commands. . First, the old *Q-Learning* extension was refactored to use BURLAP, which is a Java library that provides several reinforcement learning algorithms. Subsequently, the extension was expanded, incorporating the SARSA (λ) and Actor-Critic algorithms. To validate the operation of the extension, experiments were performed using the Cliff Walking scenario, which is a classic reinforcement learning problem. Based on the simulation runs, the results of the learned optimal policies were collected. It was verified that the learned policies remained consistent, evidencing the absence of failures in the refactoring. In addition, the new incorporated algorithms presented satisfactory results of the learned optimal policies, as expected. The developed extension is available online for use by the community, along with its respective documentation and usage examples.

Keywords: Agent-Based Simulations. Reinforcement Learning. Intelligent Agents. BURLAP. NetLogo

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

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, a plataforma NetLogo, Repast e MASON. Essas plataformas frequentemente adotam uma linguagem própria de programação. A 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 os valores de aprendizagem. 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). Segundo o estudo de Bazzanella e Santos (2021), foi evidenciado que essa extensão simplifica o desenvolvimento de simulações.

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 oferece 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. Por outro lado, a NetLogo, que é a plataforma utilizada para o desenvolvimento de

simulações, trabalha com sua própria linguagem. Dessa forma, este trabalho descreve a refatoração da 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, ou seja, 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* (FUKAYA; NAGATA, 1991).

Dessa forma, percebeu-se a necessidade de expandir a extensão *Q-Learning* existente de modo a possibilitar aos desenvolvedores 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* (λ) 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) Avaliar 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, foi realizado um estudo sobre as técnicas de aprendizagem por reforço. O estudo, feito a partir de artigos acadêmicos, visa compreender o funcionamento dos algoritmos e a sua popularidade. Depois, foram selecionados quais algoritmos serão disponibilizados pela extensão, e foi estudado sobre o funcionamento de cada algoritmo individualmente.

Foi realizado um estudo prévio da biblioteca Java BURLAP, seu funcionamento e quais artefatos disponibiliza, para que fosse possível colocar em prática a utilização da biblioteca durante a refatoração e expansão da extensão.

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

Após o desenvolvimento, foi feita uma avaliação do funcionamento da extensão. Este processo consiste em testar a implementação desenvolvida em algum cenário de simulação. Para isso, foram implementadas duas versões de uma mesma simulação para cada algoritmo disponibilizado: uma versão implementando o algoritmo diretamente na BURLAP e outra versão utilizando a extensão, com o intuito de 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).

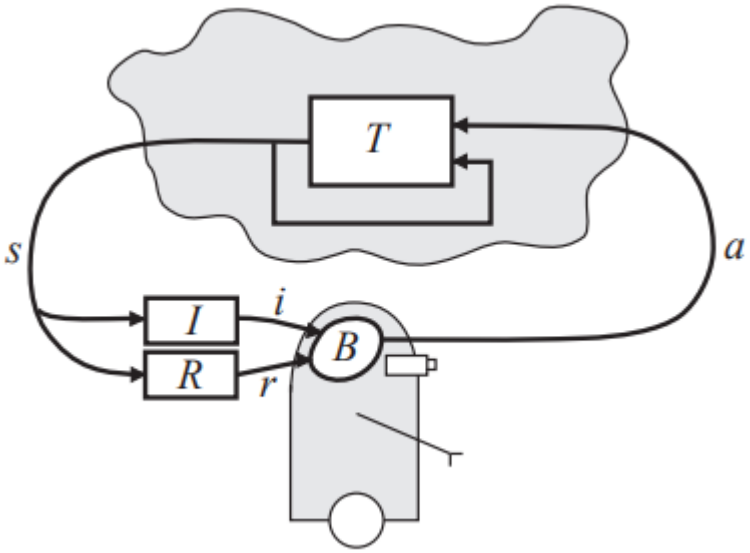
Ademais, a RL também caracteriza-se por ser orientada à objetivo, ou seja, o agente não precisa conhecer detalhes da modelagem do ambiente, ele simplesmente age neste ambiente desconhecido tentando alcançar um objetivo (SERRA et al., 2004).

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). É um algoritmo para estabelecer autonomamente uma política de ações de maneira iterativa. É possível demonstrar que o algoritmo *Q-Learning* converge para uma política ótima, 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, também conhecida por tabela Q, ou *Q-Table* (MONTEIRO; RIBEIRO, 2004).

O *Q-Learning* é um método de DT que, em vez de aprender a utilidade de um estado, aprende uma representação de ação-valor. Utiliza-se $Q(s,a)$ para representar o valor da execução da ação a no estado s. Os valores 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 demonstra a função 2.1 (RUSSEL; NORVIG, 2004).

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

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 π, e $\pi(s)$ é a ação recomendada pela política π no estado s. Um política ótima, π^*, 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, γ é um fator de desconto que descreve a preferência do agente por recompensas atuais sobre recompensas futuras. Enquanto $\mathrm{T}(s,a,s')$ 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'.

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

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

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

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 em quando, com uma pequena probabilidade ε, 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).

O pseudocódigo da Figura 2 demonstra o passo a passo do funcionamento do *Q-Learning*. O agente inicia conhecendo o conjunto de estados S e o conjunto de ações A do problema, bem como a taxa de aprendizagem α e o fator de desconto γ, ambos com valores entre 0 e 1. O algoritmo é iniciado, e para cada episódio, é definido o estado inicial, em seguida é repetido o processo de aprendizado do agente até finalizar o episódio.

Figura 2 – Algoritmo *Q-Learning*

```
   Entrada: S, A, α ∈ (0,1), γ ∈ (0,1)
 1 início
 2     para cada episodio faça
 3         s ← estado_inicial
 4         repita
 5             escolher uma ação a para o estado s
 6             executar a ação a
 7             observar o novo estado s' e a recompensa recebida R(s,a)
 8             Q(s,a) ← Q(s,a) + α(R(s,a) + γmax_a Q(s',a) − Q(s,a))
 9             s ← s'
10         até s ≠ terminal;
11     fim
12 fim
```

Fonte: Russel e Norvig (2004).

Durante o processo de aprendizado do episódio, o agente deve escolher uma ação a com base no estado s em que ele se encontra. Depois ele deve executar a ação a que foi escolhida com base em algum método de seleção da próxima ação. Após executá-la, ele deve observar o novo estado s' em que ele se encontra e resgatar sua respectiva recompensa $R(s,a)$. O cálculo de atualização do valor Q leva em consideração o seu valor antigo $Q(s,a)$, a taxa de aprendizagem α, a recompensa $R(s,a)$, o fator de desconto γ e o valor Q da ação com maior valor do estado seguinte. Por fim, é atualizado o estado atual. Esse processo se repete até o estado atual alcançar um estado terminal, quando isso ocorre, o episódio é finalizado e iniciado novamente.

2.2.2 *SARSA* (λ)

O algoritmo *SARSA*(λ) é uma modificação do algoritmo *Q-Learning* que utiliza um mecanismo de iteração de política, isto é, ele é um algoritmo *policy-based*, enquanto o *Q-Learning* é um algoritmo *value-based* (MONTEIRO; RIBEIRO, 2004). A principal diferença entre os al-

goritmos *policy-based* e *value-based* é como a regra de atualização se relaciona com a política sendo aprendida pelo agente (NISSEN, 2007). Eles convergem para diferentes políticas ótimas. O *SARSA*(λ) convergirá para uma solução ótima sob a suposição de que continuará seguindo a mesma política usada para gerar a experiência, ou seja, a mesma política do seu processo de aprendizagem. 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).

Além disso, o *SARSA*(λ) é uma adaptação do algoritmo *SARSA* combinado à traços de elegibilidade. Introduzido por Klopf (1972), o conceito de traço de elegibilidade é memorizar, em um curto tempo, os estados visitados anteriormente. Geralmente, o traço de elegibilidade decai exponencialmente levando em consideração o valor lambda λ e o fator de desconto γ, representado na função 2.4 (LAMPERTI; NEPOMUCENO; OTTONI, 2013).

$$e(s,a) \leftarrow \gamma\lambda e(s,a) \tag{2.4}$$

Assim como em muitos cenários reais, a recompensa pode ser recebida com um atraso em relação à ação executada, podendo ser benéfico possuir conhecimento em relação aos estados que contribuíram para o resultado. Nesse sentido, os traços de elegibilidade representam um fator de atenuação, para definir o atraso do sinal de reforço. A implementação dos traços pode ser feita de duas formas: traços acumulativos e traços substitutivos (GONÇALVES, 2016). O *SARSA* (λ) considera traços acumulativos, esse acúmulo de traços é definido pela função 2.5

$$e(s,a) \leftarrow e(s,a) + 1 \tag{2.5}$$

Da mesma forma que o *Q-Learning*, o *SARSA*(λ) também possui uma matriz de transição estado-ação, chamada *Q-Table*, cuja função de atualização do valores Q está representado pela função 2.6, e leva em consideração o seu valor atual, a taxa de aprendizado α, o valor de δ e a taxa de elegibilidade $e(s,a)$.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a) \tag{2.6}$$

A cada passo do processo de aprendizagem o valor de δ é atualizado utilizando a função 2.7.

$$\delta \leftarrow R(s,a) + \gamma Q(s',a') - Q(s,a) \tag{2.7}$$

O pseudocódigo da Figura 3 demonstra o passo a passo do funcionamento do *SARSA* (λ). O agente inicia conhecendo o conjunto de estados S e o conjunto de ações A do problema, bem

como a taxa de aprendizagem α, o fator de desconto γ e a taxa de elegibilidade λ, ambos com valores entre 0 e 1. O algoritmo é iniciado, e para cada episódio, é definido o estado inicial e o valor de elegibilidade, em seguida é repetido o processo de aprendizado do agente até finalizar o episódio.

Figura 3 – Algoritmo *SARSA* (λ)

Entrada: S, A, $\alpha \in (0,1)$, $\gamma \in (0,1)$, $\lambda \in (0,1)$
início
  para cada *episodio* **faça**
    $s \leftarrow estado\_inicial$
    $e(s,a) \leftarrow e(s,a) + 1$
    repita
      escolher uma ação a para o estado s
      executar a ação a
      observar o novo estado s' e a recompensa recebida $R(s,a)$
      $\delta \leftarrow R(s,a) + \gamma Q(s',a') - Q(s,a)$
      para cada (s,a) **faça**
        $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
        $e(s,a) \leftarrow \gamma\lambda e(s,a)$
      fim
      $s \leftarrow s'$
      $a \leftarrow a'$
    até $s \neq terminal$;
  fim
fim

Fonte: Sutton e Barto (2018).

Durante o processo de aprendizado do episódio, o agente deve escolher uma ação a com base no estado s em que ele se encontra. Depois ele deve executar a ação a que foi escolhida com base em algum método de seleção da próxima ação. Após executá-la, ele deve observar o novo estado s' em que ele se encontra e resgatar sua respectiva recompensa $R(s,a)$.

O cálculo de atualização do valor Q leva em consideração o seu valor antigo $Q(s,a)$, a taxa de aprendizagem α, o δ e a taxa de elegibilidade $e(s,a)$. Por conseguinte, a taxa de elegibilidade também é atualizada. E então o estado e a ação são atualizados e esse processo se repete até o estado atual alcançar um estado terminal, quando isso ocorre, o episódio é finalizado e iniciado novamente.

2.2.3 *Actor-Critic*

Nos métodos *policy-based*, como o *SARSA* (λ), é construída uma representação de uma política que é mantida na memória durante o aprendizado. Contudo, 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).

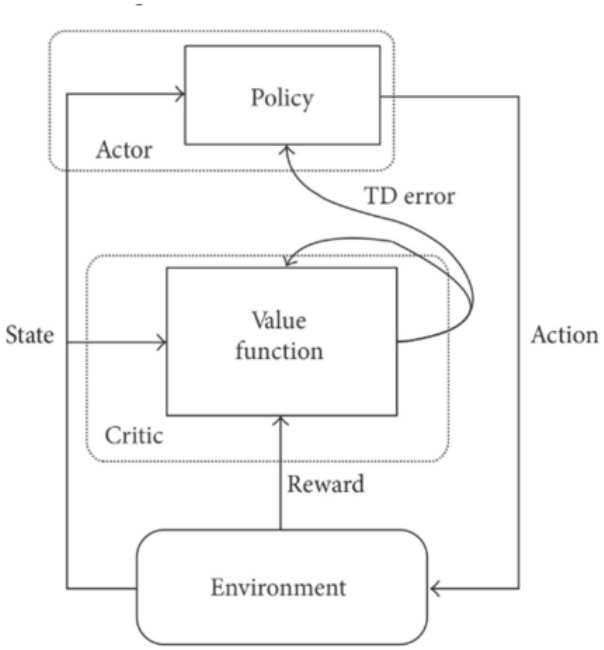
Os métodos *value-based* se limitam a um espaço de ações finito. Já os métodos *policy-based*, apesar de possibilitarem um espaço de ações contínuo, precisam ainda executar todo o episódio para conseguir alguma aprendizagem. Isto faz com que haja uma alta variância e muitas vezes ocasiona lentidão na aprendizagem (CALLOWAY, 2019).

Com o objetivo de superar os problemas que surgiram ao utilizar esses dois métodos individualmente, surgiu o algoritmo *Actor-Critic* (WILLIAMS; BAIRD, 1990), 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, ou seja, avaliar os dados que foram aprendidos. Enquanto a etapa de melhoria da política serve para melhorar a política em todas as etapas até a convergência (PETERS; SCHAAL, 2008). Em outras palavras, no algoritmo *Actor-Critic*, a etapa de avaliação pode ser interpretada como um elemento Ator que realiza a aproximação da função política, *policy-based*. E a etapa de melhoria pode ser interpretada como um elemento Crítico que realiza a aproximação da função valor, *value-based* (WANG; CHENG; YI, 2007).

Conforme Caarls (2021), o *Actor-Critic* possui uma estrutura de memória específica para armazenar a política da função de valor independentemente. Essa estrutura referente à política é o Ator, porque é utilizada para selecionar ações. E a função de valor é conhecida como Crítico, porque avalia as ações realizadas pelo Ator.

Como pode-se observar, a arquitetura do *Actor-Critic*, representada pela Figura 4, estende o modelo padrão da arquitetura RL (representado previamente na Figura 1), adicionando apenas o elemento de função de valor e o elemento de política.

Figura 4 – Arquitetura *Actor-Critic*



Fonte: Drechsler et al. (2019).

Nesta Figura, o Crítico assume a forma de um erro de DT, a única saída emitida pelo Crítico, que direciona o aprendizado tanto do Ator quanto do Crítico. Após seleção de uma determinada ação, o Crítico avalia o novo estado para determinar se houve melhora ou não. Essa avaliação é denominada como o erro DT, que é representada pelo símbolo δ, cujo valor é dado pela função 2.8 (CAARLS, 2021).

$$\delta = R(s,a) + \gamma V(s') - V(s) \tag{2.8}$$

Nesta função, V representa a função de valor implementada pelo Crítico. Se o erro DT for positivo, significa que o comportamento deve ser incentivado. Porém se o erro DT for negativo, significa que o comportamento deve ser desestimulado (CAARLS, 2021). O passo a passo do funcionamento do algoritmo está descrito no pseudocódigo apresentado na Figura 5. Vale ressaltar que podem haver diferentes implementações para os Atores e os Críticos, o algoritmo da Figura 5 refere-se a implementação encontrada na BURLAP, que é a biblioteca que será utilizada no presente trabalho.

Figura 5 – Algoritmo *Actor Critic*

Entrada: S, A, $\alpha \in (0,1)$, $\gamma \in (0,1)$, $\lambda \in (0,1)$
início
  para cada *episodio* **faça**
    $s \leftarrow estado\_inicial$
    $e(s,a) \leftarrow 1$
    repita
      ator deve escolher uma ação a para o estado s considerando preferência p
      executar a ação a
      observar o novo estado s' e a recompensa recebida $R(s,a)$
      crítico avalia
      $\delta \leftarrow R(s,a) + \gamma V(s') - V(s)$
      $V(s) \leftarrow V(s) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow e(s,a)\lambda\gamma$
      ator deve atualizar preferência
      $p \leftarrow \alpha\delta$
      $s \leftarrow s'$
    até $s \neq terminal$;
  fim
fim

Fonte: Elaborada pela Autora (2022).

O agente inicia conhecendo o conjunto de estados S e o conjunto de ações A do problema, bem como a taxa de aprendizagem α, o fator de desconto γ e a taxa de elegibilidade λ, ambos com valores entre 0 e 1. O algoritmo é iniciado, e para cada episódio, é definido o estado inicial e o valor de elegibilidade inicial, e em seguida é repetido o processo de aprendizado do agente até finalizar o episódio.

Em primeiro momento, o elemento Ator é responsável por escolher uma ação a, com base no estado s e a preferência da ação p. Depois é deve ser executada a ação a e deve-se observar o novo estado s' em que ele se encontra e resgatar sua respectiva recompensa $R(s,a)$. Após isso, o Crítico é responsável por atualizar o valor de δ, bem como os valores relacionados ao cálculo dos traços de elegibilidade e o valor de utilidade do estado $V(s)$.

Em seguida, o Ator atualiza o valor da preferência de uma ação p, levando em consideração a taxa de aprendizagem α e o valor de δ dado pelo Crítico. Por fim, o estado é atualizado e esse processo se repete até o estado atual alcançar um estado terminal, quando isso ocorre, o episódio é finalizado e iniciado novamente.

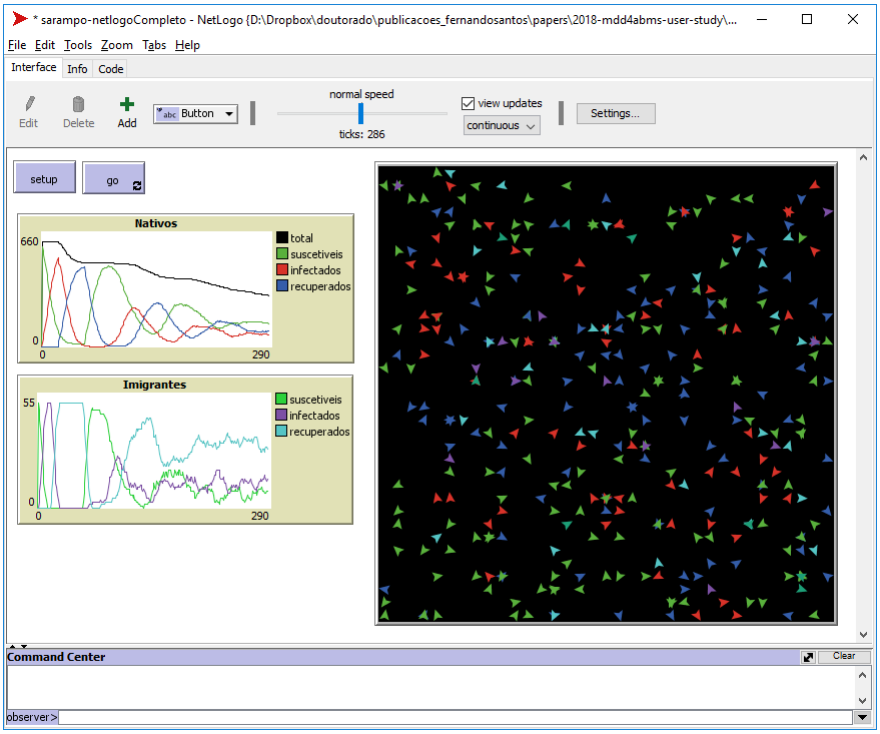
2.3 PLATAFORMA NETLOGO

A plataforma 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*. Ela 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 simulações afim de explorar os comportamentos dos agentes sob diversas condições. A NetLogo é também simples o suficiente para possibilitar que estudantes e pesquisadores criem suas próprias simulações (TISUE; WILENSKY, 2004).

A NetLogo é composta por uma tela bidimensional, como demonstra a Figura 6, chamada de mundo, na qual os agentes, chamados de `turtles`, são apresentados. O desenvolvedor pode alterar o tamanho do ambiente, que é formado por quadrados chamados de `patches`. Isto é, a unidade básica da simulação é a `turtle`, que é o agente. Cada `turtle` reside em um `patch`, sendo que várias `turtles` podem ser localizadas em um único *patch*. Todos os movimentos das *turtles* são calculados em termos de *patches* (SKLAR, 2007).

Figura 6 – Interface da plataforma NetLogo



Fonte: Santos, Nunes e Bazzan (2018).

Além disso, o desenvolvedor pode definir e instanciar seu próprio tipo de agentes, que na NetLogo é chamado de `breed`, além de usar o agente genérico `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, sendo que neste último caso deve ser informado o identificador do agente desejado (SKLAR, 2007).

Os desenvolvedores de simulações em NetLogo costumam organizar o código fonte da simulação em dois procedimentos principais: `setup` e `go`. O procedimento `setup` é responsável por inicializar a simulação. E o procedimento `go` é responsável por executar 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). A API disponibiliza diversos elementos para o desenvolvedor poder utilizar durante a implementação, facilitando a integração com a linguagem NetLogo.

Para realizar a criação de um novo comando, por exemplo, é necessário criar uma classe que implemente a interface `Command`, e se for um comando que retorne alguma informação, deve ser implementada a interface `Reporter`. Ambas as interfaces obrigarão a implementação de dois procedimentos: um que define a sintaxe do comando e outro que executa o processamento do comando. O procedimento responsável pelo processamento do comando recebe dois parâmetros, um do tipo `Argument` e outro do tipo `Context`. O parâmetro do tipo `Argument` traz as informações passadas junto ao comando, como por exemplo, algum valor númerico. Já o parâmetro do tipo `Context` traz as informações referentes ao contexto da simulação naquele instante.

A API também disponibiliza um elemento responsável por armazenar todas as informações referentes ao agente, o `Agent`, que salva informações como seu id, tamanho e forma. Além disso, quando é necessário adicionar na extensão algum procedimento que esteja na simulação, a API entende como sendo um `AnonymousCommand` ou um `AnonymousReporter` quando se trata de um procedimento com retorno, ao usar esse tipo, é possível solicitar a execução dos procedimentos pela extensão.

Para que a extensão reconheça os comandos que foram criados e quais serão suas respectivas nomenclaturas, é necessário uma classe que extenda a classe `DefaultClassManager`. Além disso, qualquer exceção pode ser tratada usando os tipos da API, como por exemplo o

`ExtensionException` e o `AgentException`. Todas essas informações estão disponíveis online na documentação da API do NetLogo[1].

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

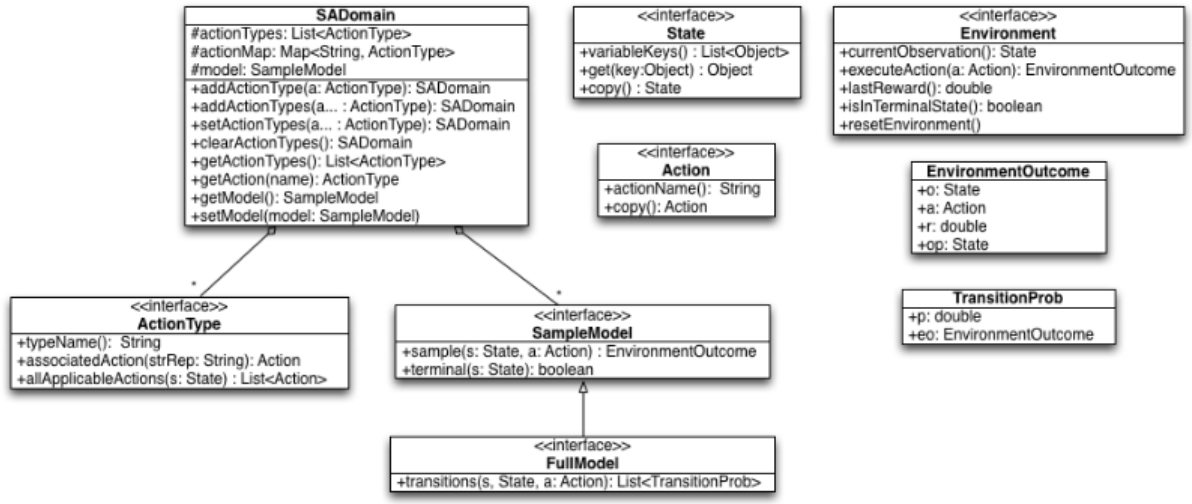
Entre as vantagens de utilizar a BURLAP, é possível mencionar que ela possui algoritmos que variam desde o planejamento clássico até o planejamento estocástico e algoritmos de aprendizado. A BURLAP também conta com ferramentas para definir visualizações de estados, episódios, funções de valor e políticas.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

A BURLAP está disponível online[2], e para conseguir utilizá-la, primeiramente é necessário se familiarizar com as interfaces Java e estruturas de dados que a biblioteca disponibiliza, apresentadas na Figura 7.

A classe `SADomain` é uma estrutura de dados responsável por armazenar as informações de como será realizada a recompensa, a definição de estado terminal e a definição de estados. A interface chamada `State`, por sua vez, serve para definir as variáveis de estado, ou seja, uma instância de um objeto que implementa essa interface 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.

[1] https://ccl.northwestern.edu/netlogo/docs/scaladoc/org/nlogo/index.html
[2] http://burlap.cs.brown.edu/

Figura 7 – Diagrama UML das interfaces/classes Java da BURLAP



Fonte: MacGlashan (2016).

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 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 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.

Hjorth et al. (2020) demonstram 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 apresentam 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 NetLogo com um ambiente Python.

Python é uma linguagem de uso geral e é cada vez mais usada para computação científica pois oferece uma variedade de bibliotecas que podem dar suporte ao desenvolvimento e teste de modelos baseados em agentes. 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 biblioteca 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 disponíveis na ferramenta do NetLogo. Para exemplificar as análises mais complexas que são habilitadas por uma interface Python, foi utilizada a biblioteca SALib Python para uma aná-

lise 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 de sistemas multiagentes baseados em quadrantes, 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, isto é, o comportamento 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. Para a área da RL, existe a Extensão *Q-Learning*, previamente desenvolvida por Kons (2019), na linguagem Scala. Essa extensão se responsabiliza por incorporar o algoritmo *Q-Learning* ao comportamento dos agentes. O estudo de Bazzanella e Santos (2021), apresentou evidências de que, ao utilizar a Extensão *Q-Learning*, houve redução na quantidade de linhas de código das simulações.

Tendo em vista os benefícios ao se utilizar a extensão, o presente trabalho propôs-se a refatorar e expandir a Extensão *Q-Learning*, para utilizar a BURLAP e abranger mais algoritmos da aprendizagem por reforço, como o *SARSA* (λ) e o *Actor-Critic*. A escolha da linguagem levou em consideração o fato de que a API da NetLogo está disponível para ser utilizada nas linguagens Java, Scala e Kotlin. Dentre essas, a única que possui uma biblioteca de aprendizagem por reforço era a linguagem Java.

A extensão foi refatorada utilizando-se a linguagem Java, e seu objetivo é disponibilizar comandos NetLogo para que o desenvolvedor possa especificar o algoritmo que deseja utilizar para o aprendizado, as variáveis que compõem 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.

Em outras palavras, a extensão serve como uma ponte entre a simulação no NetLogo e a biblioteca Java BURLAP. Portanto, ela utiliza recursos disponibilizados pela API do NetLogo para facilitar a comunicação. Ao fazer a comunicação com o NetLogo, a extensão consegue armazenar as informações que foram passadas e usar para instanciar os elementos da BURLAP.

As classes que compõem o desenvolvimento da extensão estão representadas no diagrama de classes da Figura 8. Os elementos em azul são aqueles disponibilizados pela BURLAP, no caso as interfaces a serem implementadas. Já as classes representadas na cor vermelha, são as classes disponibilizadas pela API do NetLogo. As classes da extensão, na cor branca, são responsáveis 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, ela extende a classe `DefaultClassManager` disponibilizada pela API do 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. Da

Figura 8 – Diagrama de Classes

Fonte: Elaborada pela Autora (2022).

mesma forma, todos os demais comandos, que não estão representados no diagrama, seguem a mesma lógica de ter uma classe que implementa a classe `Command`. Nos casos em que o comando tem algum tipo de retorno, é implementada a classe `Reporter`.

A classe `Session` é um *singleton*, que armazena a instância da extensão e os agentes envolvidos, que são objetos do tipo AgentLearning. Já a classe `AgentLearning`, é responsável por todas as informações necessárias para o aprendizado, como seu valor de desconto, taxa de aprendizagem, função de recompensa, informações do próprio objeto `Agent` disponibilizado pela API do NetLogo, e demais informações.

A classe `StateDefinition`, por sua vez, é reponsável por realizar a definição dos estados. Nela ficam armazenadas as informações que serão consideradas na formação de um estado. Enquanto a classe `ActionSelection` é responsável pela seleção da próxima ação, ou seja, ela armazena as informações referentes ao método de seleção da próxima ação e seus devidos parâmetros. Tanto a classe `StateDefinition` quanto a `ActionSelection`, estão agregadas à `AgentLearning`.

Para o gerenciamento das recompensas, foi desenvolvida a classe `Reward`, que implementa a interface `RewardFunction`. Ela é responsável por executar o procedimento que retorna a recompensa do agente de acordo com o estado em que ele se encontra, para isso, a classe utiliza recursos do tipo `Argument` e do tipo `Context` disponibilizados pela API do NetLogo.

Com relação aos episódios, foi desenvolvida a classe `IsEndEpisode`, que implementa a interface `TerminalFunction` disponibilizada pela BURLAP. Essa classe é responsável por executar o procedimento que retorna se um estado é terminal, isto é, que encerra o episódio. Para isso, também utiliza os recursos disponibilizados pela API do NetLogo, como o `Argument` e `Context`.

A classe `AgentState` implementa a interface `MutableState` disponibilizada pela BURLAP, e é responsável por converter as informações sobre os estados, contidas no objeto `StateDefinition`, para utilizar na BURLAP. E a classe `AgentStateModel`, por sua vez, implementa a interface `FullStateModel`, disponibilizada pela BURLAP, e é responsável por fazer a transição de estados, ou seja, de executar a ação que foi selecionada e atualizar o estado da simulação.

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. Ambas implementam a interface `DomainGenerator` e possuem uma única instância. Possuem dois métodos principais, um responsável por configurar a aprendizagem e outro responsável por executar o processo de aprendizagem.

3.1 NOVOS COMANDOS

Tendo em vista que a extensão foi ampliada para incorporar outros algoritmos além do *Q-Learning*, foi necessário ajustar o nome da extensão, que é usada na importação e usada para chamar os respectivos comandos. Anteriormente a extensão se chamava `qlearningextension`, e agora se chama `learningextension`. Além disso, para a incorporação desses novos algoritmos, foi necessário a criação de alguns comandos adicionais para serem chamados no procedimento `setup`.

Com a expansão dos novos algoritmos, surgiu a necessidade de um comando para definir qual algoritmo o desenvolvedor gostaria de incorporar no comportamento do agente. Para isso, foi criado um comando na linguagem NetLogo que recebe como entrada um valor textual, que define qual algoritmo será utilizado, se receber o valor `"qlearning"`, indica que será utilizado o *Q-Learning*, o valor `"sarsa-lambda"` representa o *SARSA* (λ) e o valor `"actor-critic"` indica que será utilizado o *Actor-Critic*. A Figura 9, linha 1, apresenta um exemplo de como utilizar esse comando para utilizar o *SARSA* (λ).

Figura 9 – Sintaxe dos comandos

```
1 learningextension:define-algorithm "sarsa-lambda"
2 learningextension:lambda 0.7
3 learningextension:setup
```

Fonte: Elaborada pela Autora (2022).

O parâmetro λ, que define a taxa de eligibilidade, não fazia parte do algoritmo *Q-Learning*, mas tanto o *SARSA* (λ), quanto o *Actor-Critic* necessitam desse parâmetro. Portanto, foi implementado um novo comando, que recebe como entrada um valor entre 0 e 1, sua sintaxe é apresentada na Figura 9, linha 2.

Como mencionado anteriormente, esses comandos apenas servem para armazenar e organizar as informações, como a intenção era utilizar a BURLAP para a aprendizagem, foi necessário a criação de um comando para, após tudo ter sido informado, instanciar os objetos necessários da BURLAP internamente na extensão e deixá-la pronta para depois só precisar executar o processo de aprendizagem. A Figura 9, linha 3, mostra como utilizar esse comando, que não possui nenhum valor de entrada.

A documentação de todos os comandos disponibilizados pela extensão está disponível online[1] e apresenta uma breve explicação sobre o que é o comando, o que ele faz, e a sua respectiva sintaxe. Além disso, no apêndice está disponível um exemplo de utilização da extensão, nele é possível observar como utilizar os novos comandos.

[1] https://github.com/agentbasedsimulations/qlearning-netlogo-extension/tree/burlap-migration

3.2 FUNCIONAMENTO

Os comandos podem ser classificados como sendo de configuração da aprendizagem e como sendo de execução do processo de aprendizado. A maioria dos comandos estão englobados na primeira categoria e são usados no processo de *setup* da simulação. A Figura 10 apresenta um trecho de código de como utilizar os comandos fornecidos pela extensão.

Figura 10 – Exemplo de trecho de código utilizando a extensão

```
1  to setup
2      ask turtles [
3          learningextension:state-def ["xcor" "ycor"]
4          (learningextension:actions [goUp] [goDown] [goLeft] [goRight])
5          learningextension:reward [rewardFunc]
6          learningextension:end-episode [isEndState] resetEpisode
7          learningextension:action-selection "e-greedy" [0.7 0.995]
8          learningextension:learning-rate 0.5
9          learningextension:discount-factor 0.5
10         learningextension:lambda 1
11         learningextension:define-algorithm "sarsa-lambda"
12         learningextension:setup
13     ]
14 end
15
16 to go
17     ask turtles [
18         learningextension:learning
19     ]
20 end
```

Fonte: Elaborada pela Autora (2022).

Como mencionado, os comandos podem ser classificados como sendo um comando de configuração (como o `state-def`), um comando que conclui a etapa de configuração (que é o `setup`), e o comando responsável por realizar a aprendizagem (que é o comando `learning`). O diagrama de atividades representado pela Figura 11 visa demonstrar o fluxo que ocorre ao utilizar esses comandos. Para tanto, existem 3 pontos de início, cada um corresponde à chamada de um comando.

O ponto de início A, representa a chamada do comando `learningextension:state-def` foi chamado, ou seja, um comando de configuração de aprendizado. A classe `StateDefinitionCommand` se responsabiliza por validar a sintaxe do comando e executa o processo do comando, utilizando-se da API do NetLogo. Nessa classe são definidas as variáveis que serão consideradas para formar o estado. Depois de validada a sintaxe, é iniciado o processamento do comando. Durante o processamento é instanciada a classe `StateDefinition` com as informações recebidas e este objeto é setado no objeto `AgentLearning`, que também

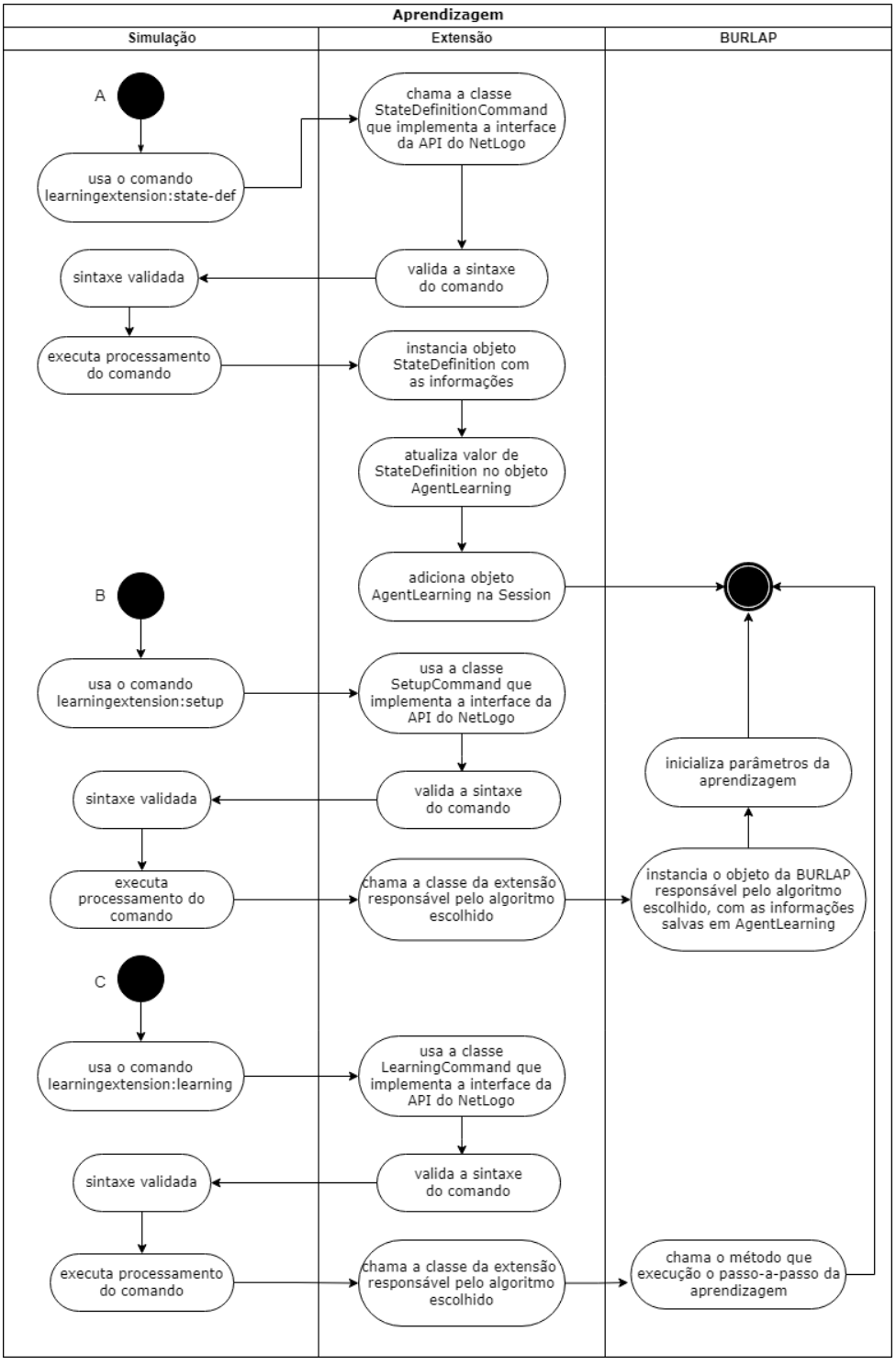
foi instanciado. A classe `AgentLearning` centraliza todas as informações relevantes para o aprendizado do agente.

Dessa mesma forma, todos os demais comandos de configuração seguem a mesma lógica de receber a informação por meio de uma classe que implementa a API do NetLogo e salva essas informações nos objetos da extensão.

Ao definir todas as configurações de aprendizado, é necessário utilizar o comando `learningextension:setup`, apresentado pelo ponto de início B. Ao ser executado, chama a classe `LearningCommand` que verifica a sintaxe e usa todas as informações que estão salvas no objeto `AgentLearning` para chamar a classe de *learning*. Essa classe varia conforme o algoritmo que foi selecionado, podendo ser o `QLearningAlgorithm`, `SarsaAlgorithm` ou `ActorCriticAlgorithm`. Nessa classe que é feita a inicialização do aprendizado na BURLAP, ou seja, a modelagem do problema.

Por fim, o ponto de início C representa a chamada do comando `learningextension:learning` que é um comando de execução do processo de aprendizado. Este comando deve ser chamado dentro de um procedimento `go` na simulação. Ao utilizá-lo, é feita a chamada para a classe `LearningCommand`, que valida a sintaxe e reconhece o algoritmo que está sendo utilizado. Ao saber o algoritmo que está sendo utilizado, chama a classe de *learning* e executa os passos do aprendizado.

Figura 11 – Diagrama de Atividades da Extensão



Fonte: Elaborada pela Autora (2022).

4 EXPERIMENTOS

Esta seção apresenta uma explicação sobre o cenário da simulação que foi utilizada para a avaliação do funcionamento dos algoritmos, bem como os resultados obtidos das execuções.

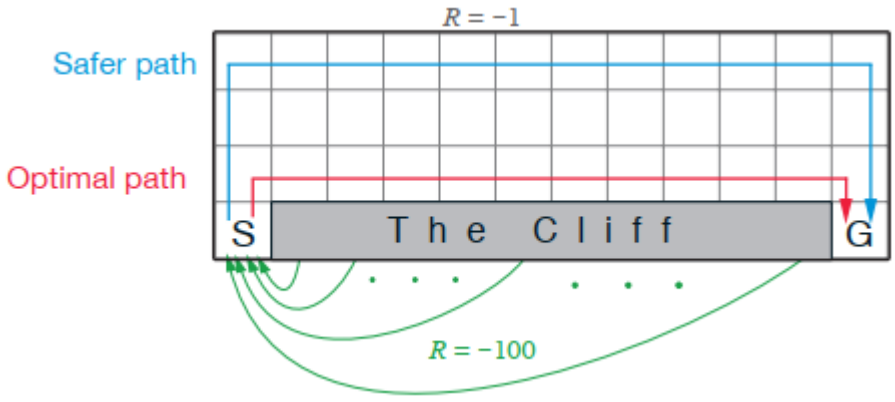
4.1 CLIFF WALKING

Com o intuito de avaliar o funcionamento da extensão, foi realizada a implementação de uma simulação na plataforma NetLogo utilizando a extensão. A simulação faz uso de um cenário clássico de aprendizagem por reforço, o *Cliff Walking*.

Neste cenário o ambiente é um mundo bidimensional dividido em células, como demonstra a Figura 12. Foi implementado um cenário com 3 células de largura e 6 células de comprimento. O agente sempre ocupa uma única célula e pode se locomover por entre as células com as ações de ir para cima, ir para baixo, ir para a esquerda e para a direita. O estado do agente, portanto, é representado pela sua posição na grade de células (SUTTON; BARTO, 2018).

O objetivo do agente é partir do ponto *S*, e chegar ao ponto *G*, traçando o menor percurso possível ou o mais seguro, sem passar pelo penhasco, representado pelas células em cinza.

Figura 12 – Cenário *Cliff Walking*



Fonte: Sutton e Barto (2018).

O agente recebe uma recompensa a cada ação tomada. Quando o agente se desloca para qualquer célula que não seja o penhasco, ele recebe uma recompensa no valor -1. Porém, quando ele se desloca para o penhasco, ele recebe uma recompensa no valor -100. O episódio é finalizado toda vez que o agente alcançar o ponto *G*, ou alcançar o penhasco, e em seguida, é iniciado um novo episódio com o agente no estado inicial *S*. A linha vermelha mostrada na Figura 12 representa a política ótima que o agente pode aprender e a linha azul representa o caminho seguro que pode ser aprendido pelo agente dependendo do algoritmo a ser utilizado.

Tendo em vista que o método de seleção da próxima ação considera um percentual de aleatoriedade nas escolhas, os resultados de uma única execução não seriam suficientes para alegar uma política ótima. Para tanto, em todas as versões da simulação que foram implementadas, foram feitas 5 execuções e foram coletadas suas respectivas médias. Além disso, vale ressaltar que a quantidade de episódios adotada foi determinada a partir de testes experimentais considerando a convergência para a política ótima. Da mesma forma, os valores dos parâmetros de entrada, que influenciam muito no processo de aprendizado, foram testados até encontrar uma convergência para a política ótima.

Vale destacar ainda que a avaliação realizada não possui o intuito de comparar o desempenho dos algoritmos, o objetivo é avaliar se os algoritmos estão funcionando corretamente.

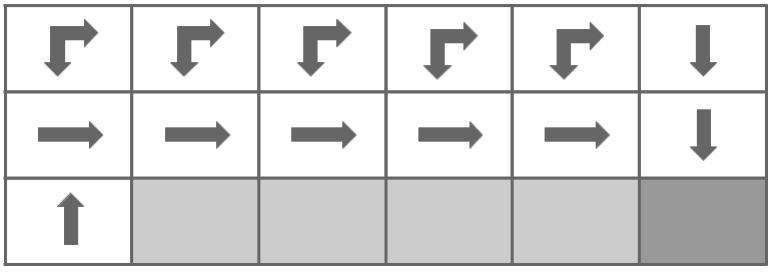
As implementações de todas as versões das simulações, bem como os resultados dessas execuções, suas respectivas médias e desvios-padrão, se encontram disponíveis online[1]. Lá também é possível encontrar uma explicação de como rodar essas simulações.

4.2 AVALIAÇÃO DO *Q-LEARNING*

Tendo em vista que o *Q-Learning* já possuía uma implementação anterior na extensão, foi necessário avaliar se a refatoração não causou implicações nos resultados das políticas aprendidas, bem como avaliar se os resultados da BURLAP permanecem consistentes após ser incorporada na extensão. Para isso, foram implementadas três versões da simulação *Cliff Walking*: uma utilizando a versão antiga da extensão, uma acessando a BURLAP diretamente (ou seja, implementada diretamente em Java, sem uso do NetLogo), e uma utilizando a extensão atual refatorada.

Para os resultados das políticas aprendidas de cada versão, foi verificado se estava coerente com a política ótima que o *Q-Learning* precisa encontrar neste cenário, que está representado na Figura 13.

Figura 13 – Política ótima do *Q-Learning*



Fonte: Elaborada pela Autora (2022).

[1] https://github.com/elobazza/rl-extension-validation

Para cada uma dessas versões, foram realizadas 5 execuções de 500 episódios. Além disso, todas as versões adotaram os mesmos valores de parâmetros: *learning-rate*: 0.1; *discount-factor*: 0.3; *epsilon*: 0.7; e *epsilon-decay*: 0.995. Ao final de cada execução, foram coletadas os valores das *Q-Tables*, que demonstram os valores das políticas aprendidas.

Com base nessas *Q-Tables*, foi feito um cálculo das médias das execuções para cada versão da simulação. A Tabela 1 apresenta os resultados das médias de execução da versão utilizando a extensão antiga e da versão utilizando diretamente a BURLAP.

A partir da *Q-Table* é possível visualizar a política aprendida para cada estado do problema. Para entender qual é a ação que mais trará benefício pro agente naquele estado, basta olhar qual das ações possui maior valor Q naquele estado. Nas próximas tabelas, este valor estará marcado em negrito para facilitar a visualização. Após visualizar os maiores valores Q para cada estado, basta verificar se a ação aprendida para um determinado estado é a mesma indicada na Figura 13 para aquele mesmo estado.

Tabela 1 – *Q-Table* média do *Q-Learning* com extensão antiga e BURLAP

	Extensão Antiga				Apenas BURLAP			
Estado	Cima	Baixo	Esquerda	Direita	Cima	Baixo	Esquerda	Direita
(0,0)	**-1.42826**	-1.42834	-1.42837	-99.84258	**-1,42826**	-1,42838	-1,42835	-99,82223
(0,1)	-1.42774	-1.42789	-1.42781	**-1.42753**	-1,42776	-1,42801	-1,42789	**-1,42753**
(0,2)	**-1,42692**	-1,42699	-1,42695	-1,42694	-1,42700	-1,42705	**-1,42698**	-1,42700
(1,1)	-1,42636	-99,01787	-1,42685	**-1,42510**	-1,42653	-99,20573	-1,42692	**-1,42510**
(1,2)	-1,42533	-1,42541	-1,42541	**-1,42530**	-1,42588	-1,42601	-1,42593	**-1,42579**
(2,1)	-1,42118	-96,79049	-1,42296	**-1,41700**	-1,42145	-95,92894	-1,42388	**-1,41700**
(2,2)	**-1,42080**	-1,42105	-1,42098	-1,42081	-1,42111	-1,42093	-1,42140	**-1,42088**
(3,1)	-1,40946	-93,98402	-1,40708	**-1,39000**	-1,40691	-94,28638	-1,40660	**-1,39000**
(3,2)	-1,40981	-1,40983	**-1,34810**	-1,40958	-1,40916	-1,40929	-1,41003	**-1,40905**
(4,1)	-1,37455	-90,68516	-1,36398	**-1,30000**	-1,34958	-91,44789	-1,37282	**-1,30000**
(4,2)	-1,38117	-1,38004	-1,38164	**-1,37980**	-1,37980	-1,37773	-1,37900	**-1,37769**
(5,1)	-1,24242	**-1,00000**	-1,24768	-1,19784	-1,25242	**-1,00000**	-1,23466	-1,20738
(5,2)	-1,29885	**-1,29588**	-1,30063	-1,29947	-1,30104	**-1,29513**	-1,30491	-1,29828

Fonte: Elaborada pela Autora (2022).

A Tabela 2, por sua vez, apresenta os resultados obtidos pelas médias das execuções da versão utilizando a extensão refatorada. É possível notar que no caso da extensão antiga, 3 estados tiveram políticas diferentes da política ótima. Já no caso da versão implementada diretamente na BURLAP, apenas 1 estado teve uma política diferente da política ótima. Enquanto na extensão refatorada, segue corretamente a política ótima, porém os valores são bastante similares, diferenciando-se apenas após algumas casas decimais.

Tabela 2 – *Q-Table* média do *Q-Learning* com extensão refatorada

	Extensão Refatorada			
Estado	Cima	Baixo	Esquerda	Direita
(0,0)	**-1,42826**	-1,42837	-1,42833	-99,88934
(0,1)	-1,42772	-1,42802	-1,42790	**-1,42753**
(0,2)	-1,42694	-1,42694	-1,42690	**-1,42684**
(1,1)	-1,42611	-98,51834	-1,42668	**-1,42510**
(1,2)	-1,42530	-1,42520	-1,42526	**-1,42512**
(2,1)	-1,42172	-96,09352	-1,42283	**-1,41700**
(2,2)	-1,42094	-1,42080	-1,42122	**-1,42067**
(3,1)	-1,40693	-91,22813	-1,40938	**-1,39000**
(3,2)	-1,40961	**-1,40914**	-1,40976	**-1,40914**
(4,1)	-1,36837	-88,26094	-1,37723	**-1,30000**
(4,2)	-1,38097	-1,37830	-1,38035	**-1,37827**
(5,1)	-1,26880	**-1,00000**	-1,28172	-1,19679
(5,2)	-1,30143	**-1,29590**	-1,30105	-1,30987

Fonte: Elaborada pela Autora (2022).

Para verificar se essa similiaridade entre os valores é um efeito causado pelos parâmetros do *Q-Learning*, foi realizado mais um experimento considerando agora apenas duas versões da simulação: uma utilizando diretamente a BURLAP e uma utilizando a extensão. Não foi utilizado a extensão antiga nesse caso, pois o experimento anterior já trazia o resultado de que não houveram implicações nos resultados das políticas aprendidas após a refatoração.

Nesse novo experimento, foram realizadas 5 execuções de 500 episódios, com os seguintes valores de parâmetros de entrada: *learning-rate*: 1; *discount-factor*: 1; *epsilon*: 0.7; e *epsilon-decay*: 0.995. Ao final de cada execução, foram coletadas os valores das *Q-Tables*, e foi realizado a média das execuções. A Tabela 3 mostra os resultados das médias das *Q-Tables*.

Tabela 3 – *Q-Table* média do *Q-Learning* com novos valores de parâmetros

	Apenas BURLAP				Extensão Refatorada			
Estado	Cima	Baixo	Esquerda	Direita	Cima	Baixo	Esquerda	Direita
(0,0)	**-7,0**	-8,0	-8,0	-100,0	**-7,0**	-8,0	-8,0	-100,0
(0,1)	-8,0	-8,0	-7,0	**-6,0**	-8,0	-8,0	-7,0	**-6,0**
(0,2)	-8,0	**-7,0**	-8,0	**-7,0**	-8,0	**-7,0**	-8,0	**-7,0**
(1,1)	-7,0	-100,0	-7,0	**-5,0**	-7,0	-100,0	-7,0	**-5,0**
(1,2)	-7,0	**-6,0**	-8,0	**-6,0**	-7,0	**-6,0**	-8,0	**-6,0**
(2,1)	-6,0	-100,0	-6,0	**-4,0**	-6,0	-100,0	-6,0	**-4,0**
(2,2)	-6,0	**-5,0**	-7,0	**-5,0**	-6,0	**-5,0**	-7,0	**-5,0**
(3,1)	-5,0	-100,0	-5,0	**-3,0**	-5,0	-100,0	-5,0	**-3,0**
(3,2)	-5,0	**-4,0**	-6,0	**-4,0**	-5,0	**-4,0**	-6,0	**-4,0**
(4,1)	-4,0	-100,0	-4,0	**-2,0**	-4,0	-100,0	-4,0	**-2,0**
(4,2)	-4,0	**-3,0**	-5,0	**-3,0**	-4,0	**-3,0**	-5,0	**-3,0**
(5,1)	-3,0	**-1,0**	-3,0	-2,0	-3,0	**-1,0**	-3,0	-2,0
(5,2)	-3,0	**-2,0**	-4,0	-3,0	-3,0	**-2,0**	-4,0	-3,0

Fonte: Elaborada pela Autora (2022).

Com base nesses resultados, é possível notar como os valores da *Q-Table* variam conforme a mudança nos valores dos parâmetros de entrada e é possível ver também que, com essa combinação de valores, chegou-se exatamente na política ótima esperada na Figura 13.

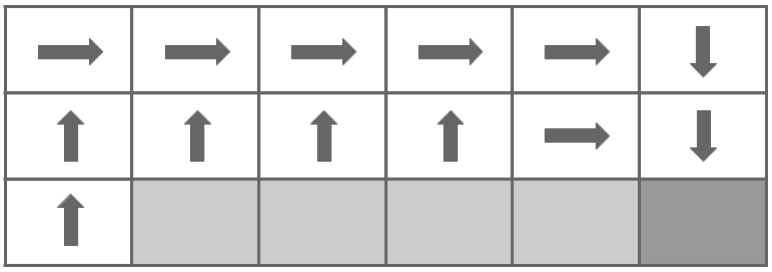
Desta forma, é possível evidenciar que a refatoração, além de não acarretar implicações no funcionamento do aprendizado, trouxe resultados satisfatórios nas políticas aprendidas pelo agente ao utilizar a BURLAP na extensão.

4.3 AVALIAÇÃO DO *SARSA* (λ)

O algoritmo *SARSA* (λ) é um algoritmo novo na extensão. Ele foi avaliado com duas versões da simulação *Cliff Walking*: uma utilizando a BURLAP diretamente, e outra utilizando a extensão. Vale lembrar que o *SARSA* (λ) aprende uma política ótima diferente do *Q-Learning* pois considera a segurança do agente como item fundamental do processo (NISSEN, 2007).

Da mesma forma que foi feito para o *Q-Learning*, para os resultados das políticas aprendidas de cada versão, foi verificado se estava coerente com a política ótima que o *SARSA* (λ) precisa encontrar neste cenário, que está representado na Figura 14.

Figura 14 – Política ótima do *SARSA* (λ)



Fonte: Elaborada pela Autora (2022).

Para cada uma dessas versões, foram realizadas 5 execuções de 700 episódios, que foi o período médio observado para a convergência para a política ótima. Além disso, todas as versões adotaram os mesmos valores de parâmetros: *learning-rate*: 0.1; *discount-factor*: 0.3; *epsilon: 0.7*; *epsilon-decay*: 0.995; e *lambda*: 0.9. Ao final de cada execução, foram coletadas os valores obtidos das *Q-Tables*.

Foi realizado o cálculo das médias das execuções para cada versão da simulação. A Tabela 4 apresenta os resultados das médias de execução da versão utilizando diretamente a BURLAP e a versão utilizando a extensão.

Tabela 4 – *Q-Table* média do *SARSA* (λ) usando direto a BURLAP e usando a extensão

	Apenas BURLAP				Utilizando Extensão			
Estado	Cima	Baixo	Esquerda	Direita	Cima	Baixo	Esquerda	Direita
(0,0)	**-1,42914**	-3,67604	-4,18387	-99,89715	**-1,42883**	-2,49786	-2,61779	-99,94464
(0,1)	**-1,42860**	-3,50119	-1,47873	-3,33155	**-1,42854**	-2,86124	-1,50374	-2,70086
(0,2)	-1,43641	-1,60121	-1,43658	**-1,42858**	-1,43215	-1,55949	-1,43108	**-1,42840**
(1,1)	-1,68418	-98,94926	**-1,57263**	-4,51620	-1,56623	-99,06203	**-1,55463**	-2,82806
(1,2)	-1,48034	-2,97629	-1,45297	**-1,42788**	-1,45816	-2,94894	-1,45925	**-1,42776**
(2,1)	**-1,49331**	-83,42055	-4,75664	-2,81698	**-1,42989**	-83,32282	-5,13225	-4,30573
(2,2)	-1,47917	-3,01613	-1,47485	**-1,42553**	-1,47435	-2,45719	-1,50654	**-1,42554**
(3,1)	**-1,42887**	-61,03089	-3,01996	-4,21420	**-1,42829**	-68,61894	-4,24472	-3,90162
(3,2)	-1,46670	-2,56292	-1,57105	**-1,41796**	-1,46664	-2,96519	-1,52609	**-1,41802**
(4,1)	-1,36158	-58,49340	-2,86170	**-1,29215**	-1,33254	-58,03221	-4,22519	**-1,29849**
(4,2)	-1,45713	-2,90284	-1,50978	**-1,39105**	-1,46306	-2,48337	-1,48219	**-1,39086**
(5,1)	-1,32168	**-1,00000**	-2,33036	-1,27947	-1,32044	**-1,00000**	-2,40547	-1,28788
(5,2)	-1,40569	**-1,30218**	-1,44315	-1,40431	-1,40320	**-1,30091**	-1,46451	-1,40077

Fonte: Elaborada pela Autora (2022).

É possível notar que no caso da versão implementada diretamente na BURLAP, apenas 1 estado teve uma política aprendida diferente da política ótima, e esse comportamento se repetiu também na versão utilizando a extensão. E é possível ver que essa política possui uma pequena diferença de valor com a política que deveria ter sido aprendida. Então, da mesma forma que foi feito com o *Q-Learning*, também foi realizado um novo experimento com uma nova combinação de valores de parâmetros de entrada.

Nesse novo experimento, foram realizadas 5 execuções de 700 episódios, com os seguintes valores de parâmetros de entrada: *learning-rate*: 0.5; *discount-factor*: 0.5; *epsilon*: 0.7; e *epsilon-decay*: 0.995. Ao final de cada execução, foram coletadas os valores das *Q-Tables*, e foi realizado a média das execuções. A Tabela 5 mostra os resultados das médias das *Q-Tables*.

Tabela 5 – *Q-Table* média do *SARSA* (λ) com novos valores de parâmetros

	Apenas BURLAP				Utilizando Extensão			
Estado	Cima	Baixo	Esquerda	Direita	Cima	Baixo	Esquerda	Direita
(0,0)	**-1,99650**	-9,98625	-11,32378	-100,0	**-1,99636**	-12,05645	-5,23845	-100,0
(0,1)	**-1,99269**	-3,81315	-2,40526	-12,20371	**-1,99270**	-2,51677	-2,24678	-15,01358
(0,2)	-2,12868	-3,40504	-2,06120	**-1,98534**	-2,30813	-2,32880	-2,01280	**-1,98531**
(1,1)	**-3,22029**	-100,0	-5,73828	-3,39884	**-1,76315**	-100,0	-7,02749	-2,57476
(1,2)	-2,06023	-3,58093	-2,05206	**-1,97068**	-3,20844	-8,05625	-2,33142	**-1,98297**
(2,1)	**-6,89403**	-99,99801	-11,87911	-13,69115	**-1,98662**	-99,99982	-12,74462	-19,45179
(2,2)	-2,17420	-4,54742	-2,16755	**-1,94136**	-2,77747	-3,91706	-2,12292	**-1,94073**
(3,1)	**-3,23898**	-99,79889	-14,85599	-12,48418	**-1,97260**	-98,12012	-9,42074	-22,32461
(3,2)	-2,10563	-3,55477	-3,22778	**-1,88271**	-2,18766	-11,85778	-2,09680	**-1,88126**
(4,1)	-2,15305	-99,76563	-2,85505	**-1,76333**	-1,90400	-99,32617	-10,46501	**-1,50151**
(4,2)	-1,91195	-4,32921	-1,92069	**-1,79497**	-1,93889	-9,60968	-1,96275	**-1,76251**
(5,1)	-1,76520	**-1,00000**	-1,87996	-1,50561	-1,78433	**-1,00000**	-7,38230	-1,50117
(5,2)	-1,78757	**-1,50527**	-1,88405	-1,78785	-1,96631	**-1,50002**	-1,90512	-1,80811

Fonte: Elaborada pela Autora (2022).

Com base nesses resultados, é possível notar que os valores da *Q-Table* apresentaram uma diferença maior de valor entre as possíveis ações em um mesmo estado e é possível ver também que, com essa combinação de valores, chegou-se exatamente na política ótima esperada na Figura 14.

Desta forma, é possível evidenciar que o algoritmo incorporado *SARSA* (λ) trouxe resultados satisfatórios nas políticas aprendidas pelo agente ao utilizar a BURLAP na extensão.

4.4 AVALIAÇÃO *ACTOR-CRITIC*

O algoritmo *Actor-Critic* também é um algoritmo novo na extensão. Da mesma maneira que o algoritmo *SARSA* (λ), foi avaliado duas versões da simulação *Cliff Walking*: uma utilizando a BURLAP diretamente, e outra utilizando a extensão. O *Actor-Critic* também considera a segurança do agente como item fundamental do processo de aprendizado, o que implica que sua política será mais parecida com a do *SARSA* (λ).

Para cada uma dessas versões, foram realizadas 5 execuções de 100 episódios, que foi o período médio observado para a convergência para a política ótima. Além disso, todas as versões adotaram os mesmos valores de parâmetros: *learning-rate*: 0.5; *discount-factor*: 1; e *lambda*: 0.3. Ao final de cada execução, foram coletadas os valores de utilidade obtidos.

Foi realizado o cálculo das médias das execuções para cada versão da simulação. A Tabela 6 apresenta os resultados das médias de execução da versão utilizando diretamente a BURLAP e a versão utilizando a extensão.

Tabela 6 – Média dos valores de Utilidade do *Actor-Critic*

	Apenas BURLAP	Utilizando Extensão
Estado	V(s)	V(s)
(0,0)	-9,08748	-34,44597
(0,1)	-8,05724	-8,00000
(0,2)	-7,03284	-7,00000
(1,1)	-30,73861	-36,99150
(1,2)	-6,05723	-6,00000
(2,1)	-35,99048	-47,34018
(2,2)	-5,00661	-5,00000
(3,1)	-29,25076	-37,33405
(3,2)	-4,00216	-4,00000
(4,1)	-23,78968	-34,25671
(4,2)	-3,00053	-3,00000
(5,1)	-1,00001	-1,00000
(5,2)	-2,25166	-2,00000

Fonte: Elaborada pela Autora (2022).

O *Actor-Critic* não apresenta seus resultados com base em uma *Q-Table*, ele demonstra em forma de valor de utilidade, como a própria RL faz, atribuir um número para expressar a

desejabilidade de um estado para o agente. Isto é, cada estado recebe um valor, e o agente segue o caminho de estados sempre tentando maximizar sua recompensa. Então, ao estar em um determinado estado, o próximo que ele selecionar é, entre seus possíveis futuros estados, o que lhe ofere uma recompensa maior.

Com base nesses resultados, é possível comparar a política aprendida pelo agente em ambas as versões da simulação. A Figura 15 apresenta os valores de utilidade ao utilizar a versão que implementa direto na BURLAP.

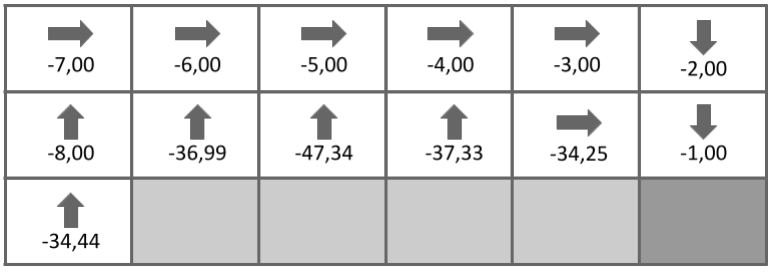
Figura 15 – Política ótima do *Actor-Critic* usando diretamente a BURLAP

Fonte: Elaborada pela Autora (2022).

As setas nas imagens representam que, ao estar num determinado estado, qual possível estado vizinho o beneficia mais. Ou seja, estando num determinado estado, qual dos estados seguintes possui maior valor de utilidade e então a flecha aponta para aquela direção.

A Figura 16, por sua vez, apresenta os valores de utilidade ao utilizar a versão que usa a extensão. E segue o mesmo esquema de representação utilizando flechas para representar a política ótima encontrada.

Figura 16 – Política ótima do *Actor-Critic* utilizando a extensão



Fonte: Elaborada pela Autora (2022).

Com base nesses resultados, é possível notar grande semelhança nos valores. Apesar de o estado inicial apresentar valores de utilidade diferentes, isso não influenciou na política encontrada. É possível visualizar também a coerência das políticas aprendidas em ambas as versões, que era exatamente a política ótima esperada.

Desta forma, é possível evidenciar que o algoritmo incorporado *Actor-Critic* trouxe resultados coerentes com o que era esperado no que diz respeito às políticas aprendidas pelo agente ao utilizar a BURLAP na extensão.

4.5 CONSIDERAÇÕES SOBRE OS RESULTADOS

Considerando os resultados apresentados, foi possível constatar que a refatoração não causou nenhuma inconsistência no aprendizado das políticas, isto é, a refatoração não causou nenhum problema na aprendizagem que já estava funcionando. E também foi possível perceber que os valores dos parâmetros de entrada influenciam nos valores das *Q-Tables*.

As primeiras combinações de valores dos parâmetros de entrada, no caso do *Q-Learning* e do *SARSA* (λ) trouxeram resultados com poucas divergências, porém com valores muito próximos. Mas ao reajustar essas combinações, os resultados foram mais satisfatórios e encontraram a política ótima esperada de cada algoritmo.

Por fim, vale mencionar ainda o alto desempenho do *Actor-Critic*. Enquanto o *Q-Learning* levou 500 episódios e o *SARSA* (λ) levou 700 episódios para a convergência, o algoritmo *Actor-Critic* levou apenas 100 episódios para encontrar sua política ótima.

5 CONCLUSÕES

O presente trabalho propôs-se a 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 em NetLogo, 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 foi o seu propósito de facilitar a criação de agentes inteligentes. Nele foi feita a refatoração do *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. Após a refatoração da extensão para utilizar a BURLAP, foram realizados experimentos utilizando o algoritmo *Q-Learning* e verificou-se que não houve implicações na política ótima aprendida. Verificou-se também que, ao utilizar uma combinação de parâmetros de entrada ideais, é possível encontrar a política ótima na extensão refatorada. Isto evidencia que a refatoração para utilizar a BURLAP não causou inconsistências na extensão.

Outro diferencial foi a incorporação de novos algoritmos de RL, o *SARSA* (λ) e o *Actor-Critic*, possibilitando ao desenvolvedor a escolha de qual algoritmo utilizar, ampliando as possibilidades de desenvolvimento. Após a implementação, também foram feitos experimentos e constantou-se que traziam resultados coerentes e satisfatórios. Inclusive, o *Actor-Critic* se destacou com seu tempo de convergência para a política ótima, por ser bem mais ágil no processo de aprendizado.

Uma sugestão para trabalhos futuros, seria a validação com outros cenários da RL, para evidenciar que a extensão refatorada pode ser aplicada em diversos cenários. Também seria interessante a incorporação de novos algoritmos para ampliar ainda mais as possibilidades do desenvolvedor, pois a BURLAP conta com diversos algoritmos que poderiam ser incorporados na extensão. E, por fim, uma última sugestão seria disponibilizar um comando para exportação dos resultados, em formato de planilha, para facilitar a coleta de dados, que foi um grande desafio encontrado no desenvolvimento deste trabalho.

APÊNDICE A - EXEMPLO DE IMPLEMENTAÇÃO

Exemplo de uso do algoritmo *SARSA* (λ) no cenário *Cliff Walking*

```
1  extensions[learningextension]
2  breed[Walkers Walker]
3  patches-own [reward]
4
5  to setup
6    clear-all
7    ask patches [set pcolor grey set reward -1 set plabel reward]
8    ask patches with [pycor = 0 and pxcor > 0 and pxcor < 5][set pcolor blue set reward -100 set plabel
         reward]
9    ask patch 0 0 [set pcolor red + 1]
10   ask patch 5 0 [set pcolor green]
11   ask patch 0 0 [sprout-walkers 1[set color yellow]]
12
13   ask Walkers [
14     learningextension:state-def ["xcor" "ycor"]
15     (learningextension:actions [goUp] [goDown] [goLeft] [goRight])
16     learningextension:reward [rewardFunc]
17     learningextension:end-episode [isEndState] resetEpisode
18     learningextension:action-selection "e-greedy" [0.7 0.995]
19     learningextension:learning-rate 0.5
20     learningextension:discount-factor 0.5
21     learningextension:lambda 1
22     learningextension:define-algorithm "sarsa-lambda"
23     learningextension:setup
24   ]
25 end
26
27 to go
28   ask Walkers [
29     set xcor 0 set ycor 0
30     while [[pcolor] of patch-here = grey or [pcolor] of patch-here = red + 1][
31       learningextension:learning
32     ]
33   ]
34 end
35
36 to-report rewardFunc
37   report [reward] of patch-here
38 end
39
40 to goUp
```

```
41    if ycor + 1 != max-pycor + 1 [
42      set heading 0
43      fd 1
44    ]
45  end
46
47  to goDown
48    if ycor - 1 != (- 1) [
49      set heading 180
50      fd 1
51    ]
52  end
53
54  to goLeft
55    if xcor - 1 != (- 1) [
56      set heading 270
57      fd 1
58    ]
59  end
60
61  to goRight
62    if xcor + 1 != max-pxcor + 1 [
63      set heading 90
64      fd 1
65    ]
66  end
67
68  to-report isEndState
69    if [pcolor] of patch-here = blue or [pcolor] of patch-here = green [
70      report true
71    ]
72    report false
73  end
74
75  to resetEpisode
76    setxy 0 0
77  end
```

REFERÊNCIAS

BAZZANELLA, E.; SANTOS, F. d. Does a q-learning netlogo extension simplify the development of agent-based simulations? **15th Workshop-School on Agents, Environments, and Applications**, 2021.

CAARLS, W. **Estudo de técnicas de aprendizado por reforço aplicadas ao controle de processos químicos**. Tese (Doutorado) — PUC-Rio, 2021.

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.

DRECHSLER, M. F. et al. **Anti-slip control with an actor-critic reinforcement learning algorithm**. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Joinville, SC, 2019.

FUKAYA, T.; NAGATA, M. An automatic debugging approach for logic programming with a method for propagating constraints. In: IEEE COMPUTER SOCIETY. **1991 The Fifteenth Annual International Computer Software & Applications Conference**. [S.l.], 1991. p. 418–419.

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.

GONÇALVES, D. V. Controle adaptativo de processo de nível utilizando aprendizado por reforço ator-crítico. 2016.

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 (λ) 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.

KLOPF, A. H. **Brain function and adaptive systems: a heterostatic theory**. [S.l.]: Air Force Cambridge Research Laboratories, Air Force Systems Command, United . . . , 1972.

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.

LAMPERTI, R. D.; NEPOMUCENO, E. G.; OTTONI, A. L. C. Aprendizado por reforco no domínio do futebol de rob^os 2d: Uma analise do traco de elegibilidade com uma comparacao entre os algoritmos q-learning (λ) e sarsa (λ). 2013.

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 (λ) 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**. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 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.

WILLIAMS, R. J.; BAIRD, L. C. A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming. In: CITESEER. **Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems**. [S.l.], 1990. p. 96–101.

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.