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

KEVIN KONS

BIBLIOTECA Q-LEARNING PARA DESENVOLVIMENTO DE SIMULAÇÕES COM AGENTES NA PLATAFORMA NETLOGO

IBIRAMA

2019

KEVIN KONS

BIBLIOTECA Q-LEARNING PARA DESENVOLVIMENTO DE SIMULAÇÕES COM AGENTES 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

2019

KEVIN KONS

BIBLIOTECA Q-LEARNING PARA DESENVOLVIMENTO DE SIMULAÇÕES COM AGENTES NA PLATAFORMA NETLOGO

Trabalho de Conclusão apresentado ao Curso de Engenharia de Software, da Universidade do Estado de Santa Catarina, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Software

Banca Examinadora

Orientador

Prof. Dr. Fernando dos Santos
UDESC/IBIRAMA

Membros:

Prof. Dr. Tiago Luiz Schmitz
UDESC/IBIRAMA

Prof. Dr. Adilson Vahldick
UDESC/IBIRAMA

Ibirama – SC, 18 de novembro de 2019

Dedico este trabalho aos meus pais que me fizeram quem sou e me deram valores.

AGRADECIMENTOS

Gostaria de agradecer primeiramente aos meus pais, que sempre me deram total apoio tanto durante esse trabalho como no período inteiro de graduação, sem eles esse trabalho não seria possível. Agradeço também a minha irmã pelo apoio e aprendizados compartilhados.

Aos meus companheiros do GoF também quero agradecer, a presença de vocês durante esse percurso foi um diferencial e tornou-o muito mais agradável, obrigado por todos os momentos e conhecimentos compartilhados.

Sou grato também aos professores que contribuiram para minha formação e que compartilharam tanto conhecimento comigo. Sou grato principalmente ao meu orientador Fernando dos Santos que foi indispensável para o desenvolvimento deste trabalho, sempre estando presente e se dispondo a ajudar.

Não poderia também deixar de agradecer a todos os amigos que fiz durante esses quatro anos de graduação, tanto dentro como fora da universidade, vocês ficarão marcados para sempre na minha vida. Agradeço em especial ao Briani, Diego, Douglas B., Euler, Felipe, Gabriel Soares, Jony, Juan, Léo Polastri, Luciano, Pedro e Rafael Tenfen.

Agradeço também a Camilla Serafim que durante esses últimos anos sempre esteve presente na minha vida, me dando apoio, incentivo e carinho. Você foi essencial nessa jornada e por isso não poderia deixar de lhe agradecer.

RESUMO

Este trabalho apresenta uma extensão para a plataforma NetLogo que possibilita aos desenvolvedores de simulações com agentes incorporarem o comportamento de aprendizagem por reforço *Q-Learning*. A extensão incorporou novos comandos ao NetLogo para que o desenvolvedor possa facilmente especificar os seguintes elementos de aprendizagem por reforço: estados, ações, recompensa, seleção de ações, cláusula de fim de episódio, reinício de episódio, taxa de aprendizagem e fator de desconto. Além disto, a extensão fornece comandos para ativar o algoritmo de aprendizagem *Q-Learning*. A partir da extensão criada, os desenvolvedores não precisam se preocupar em implementar estruturas de dados para os elementos de aprendizagem e nem o algoritmo *Q-Learning*, facilitando o desenvolvimento de simulações. A corretude da implementação da extensão foi verificada através desenvolvimento do cenário clássico de aprendizagem *cliff walking*. Neste cenário, a simulação desenvolvida com a extensão encontrou a política ótima, evidenciando a corretude da sua implementação. Para verificar a complexidade de uso da extensão, foi realizada uma comparação das características do código de uma simulação que utiliza a extensão com uma simulação implementada sem a extensão. A partir desta comparação foi possivel observar que o código de uma simulação que utilize a extensão é menos complexo. A extensão desenvolvida está disponível para uso da comunidade através do *extension manager* do NetLogo, juntamente com documentação e exemplo de utilização.

Palavras-chave: Simulações com Agentes, NetLogo, Q-Learning, Extensão.

ABSTRACT

This paper presents an extension to the NetLogo platform that enables agent-based modeling and simulation developers to incorporate reinforcement learning behavior through Q-Learning. The extension has added new commands to NetLogo so that the developer can easily specify the following reinforcement learning elements: states, actions, reward, action selection, end of episode clause, episode restart, learning rate, and discount factor. In addition, the extension provides commands to use the Q-Learning algorithm. With the use of the extension, developers do not have to worry about implementing data structures for learning elements neither the Q-Learning algorithm, easing the development of simulations. The correctness of the implementation of the extension was verified by developing the classic cliff walking learning scenario. In this scenario, the simulation developed with the extension found the optimal policy, showing the correctness of its implementation. To verify the complexity of using the extension, a comparison of the code characteristics of a simulation using the extension with a simulation implemented without the extension was performed. From this comparison it was possible to observe that the code of a simulation that uses the extension is less complex. The developed extension is available for community use through the NetLogo extension manager, along with documentation and an usage example.

Keywords: Extension, Netlogo, Q-Learning, Agent-based simulation, Library.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABMS	*Agent-Based Modeling and Simulation*
API	*Application Programmer's Interface*
DT	Diferença Temporal

SUMÁRIO

1 INTRODUÇÃO

Simulações com agentes têm sido amplamente utilizadas para entender o comportamento emergente de sistemas complexos. Esses sistemas são compostos por múltiplas entidades, ou agentes, que podem interagir entre si e estão situadas em um ambiente que podem perceber e modificar através de suas ações. Não é trivial formular modelos analíticos que possam simular o comportamento de tais sistemas complexos. Construí-los é uma tarefa desafiadora que tem sido amplamente investigada no contexto de *agent-based modeling and simulation* (ABMS), um paradigma de simulação que utiliza agentes para reproduzir e explorar um fenômeno sob investigação (KLÜGL; BAZZAN, 2012).

ABMS tem sido usado para modelar simulações em muitas áreas de aplicação, como tráfego, ecologia, economia e epidemiologia (MACAL; NORTH, 2014). De acordo com Macal e North (2014), nesses casos, o ABMS foi selecionado como paradigma de simulação porque pode incorporar explicitamente a complexidade decorrente do comportamento e das interações individuais existentes nos cenários do mundo real. O desenvolvimento de simulações com agentes envolve diferentes papéis que devem interagir e se comunicar, cada função possui especialidades distintas. Geralmente, o conhecimento do domínio concentra-se no papel do modelador, enquanto o conhecimento técnico (em simulações com agentes e suas plataformas) está concentrado nas funções de cientista da computação e programador (GALÁN et al., 2009).

Em simulações com agentes, os agentes podem incorporar diferentes capacidades para se adaptarem às mudanças em si mesmos ou no ambiente (KLÜGL; BAZZAN, 2012). Uma dessas capacidades é a aprendizagem por reforço que consiste em aprender o que fazer — como mapear situações para ações — de modo a maximizar um sinal de recompensa numérico. O agente não sabe quais ações tomar, mas precisa descobrir quais ações geram mais recompensa ao testá-las (SUTTON; BARTO, 2018). Um dos métodos de aprendizagem por reforço se chama *Q-learning*, em que o agente aprende uma representação de ação-valor. O método *Q-learning* possui uma propriedade muito importante: é livre de modelo. Deste modo ele não precisa de um modelo para aprendizagem ou seleção de ações (RUSSEL; NORVIG, 2004).

Uma das plataformas mais utilizadas para ABMS é a NetLogo. A plataforma NetLogo é um ambiente de modelagem para simular fenômenos naturais e sociais complexos. É particularmente adequada para modelar sistemas complexos que evoluem ao longo do tempo. Os programadores podem dar instruções para centenas ou milhares de agentes independentes operando simultaneamente, a fim de explorar as conexões entre os comportamentos de nível micro dos indivíduos e os padrões de nível macro que emergem de suas interações (TISUE; WILENSKY, 2004).

Apesar de ser amplamente utilizada para ABMS, atualmente não existe nenhuma biblioteca para a plataforma NetLogo que permita incorporar em um agente o comportamento de aprendizagem por reforço por meio do método *Q-learning*. Isso significa que sempre que for necessário a um agente tal comportamento o desenvolvedor deve implementá-lo por completo. Este trabalho se propõe a desenvolver uma biblioteca que elimine esse problema, de modo que o desenvolvedor de simulações com agentes em NetLogo se preocupe com aquilo que é relevante ao seu problema e não com detalhes inerentes do algoritmo *Q-learning*.

1.1 PROBLEMA

Atualmente, quando a modelagem de uma simulação com agentes na plataforma NetLogo requer incorporar a um agente o comportamento de aprendizagem por reforço por meio do método *Q-learning*, tal comportamento deve ser completamente codificado pelo desenvolvedor da simulação. Tal fato se dá pela inexistência de extensões que disponibilizem um mecanismo para incorporar o *Q-learning* na simulação.

Portanto, questiona-se a possibilidade do desenvolvimento de uma extensão para a plataforma NetLogo que permita aos desenvolvedores de simulações com agentes incorporar, de forma simples, o comportamento de aprendizagem por reforço nos agentes.

1.2 OBJETIVOS

Nesta seção estão dispostos o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Objetivo geral

Disponibilizar uma extensão para a plataforma NetLogo que permita aos desenvolvedores de simulações com agentes incorporar, com facilidade, o método de aprendizagem por reforço *Q-learning* ao comportamento dos agentes.

1.2.2 Objetivos específicos

a) Desenvolver uma extensão NetLogo que permita a incorporação do método de aprendizagem *Q-Learning* em simulações com agentes.

b) Validar a extensão proposta.

c) Submeter a extensão desenvolvida à comunidade de desenvolvedores NetLogo, para sua adoção e integração ao simulador.

1.3 JUSTIFICATIVA

A necessidade da utilização de agentes que incorporam um comportamento de aprendizagem por reforço por meio do método *Q-learning* é frequentemente enfrentada por desenvolvedores de simulações com agentes. El-Tantawy e Abdulhai (2010), por exemplo, desenvolveram um controle semafórico acíclico baseado no *Q-learning* que usa uma sequência variável de faseamento.

Definir um comportamento de aprendizagem por reforço por meio do método *Q-learning* a um agente em simulações no NetLogo pode ser uma barreira para alguns desenvolvedores. Para implementar o algoritmo é necessário conhecimentos em programação e Inteligência Artificial. Disponibilizar uma extensão que facilite incorporar tal comportamento a um agente permite que um maior número de pessoas possam fazer simulações que utilizem aprendizado, aumenta a produtividade daqueles que já fazem e impede que erros de código levem a conclusões incorretas.

1.4 HIPÓTESE

Por meio de uma extensão para a plataforma NetLogo, que permita a desenvolvedores de simulações com agentes incorporar o comportamento de aprendizagem por reforço baseado no método *Q-learning*, é possível simplificar o desenvolvimento de simulações em que os agentes utilizam tal método de aprendizado.

A extensão que será desenvolvida a partir desse estudo permitirá ao projetista focar-se nos aspectos específicos do agente que deseja simular, ao invés de gastar seu tempo com toda a implementação do comportamento de aprendizado.

1.5 METODOLOGIA

Para desenvolver a extensão de *Q-Learning* para o NetLogo, as seguintes etapas devem ser executadas.

a) Entender por completo o método *Q-Learning*: envolve um estudo aprofundado sobre o método com o intuito de compreender seus conceitos, variáveis e algoritmo.

b) Estudar a estrutura das extensões para a plataforma NetLogo: consiste em compreender como utilizar a API para criação de extensões e como incorporar uma extensão a um modelo no NetLogo.

c) Arquitetar a extensão: compreende separar o que será tarefa do projetista codificar do que será previamente implementado na extensão, além de se especificar como será a integração do código gerado pelo projetista com os códigos da extensão.

d) Implementar a extensão: implica em codificar a arquitetura proposta na etapa anterior.

e) Validar a extensão: envolve testar a implementação desenvolvida em algum cenário já conhecido para garantir seu funcionamento.

f) Submeter a extensão: fazer a extensão estar facilmente acessível através do gerenciador de extensões do NetLogo.

2 FUNDAMENTAÇÃO TEÓRICA

Para realizar o desenvolvimento deste trabalho, se faz necessário o uso de técnicas de aprendizagem por reforço e outras ferramentas disponíveis atualmente. Nesta seção há um levantamento dos itens necessários para o desenvolvimento deste trabalho.

2.1 SIMULAÇÕES COM AGENTES

Agentes podem tomar decisões a partir de deduções lógicas, modo que se assemelha ao raciocínio humano, ou até mesmo tomarem decisões através da combinação de dedução e outro mecanismo de tomada de decisão (WOOLDRIDGE, 2009). A definição comumente adotada de agente especifica o seguinte conjunto de propriedades que caracteriza uma entidade agente: (i) autonomia: a capacidade de operar sem intervenção; (ii) reatividade: a capacidade de perceber o ambiente em que está situado e responder a mudanças nele; (iii) proatividade: a capacidade de tomar a iniciativa de acordo com suas metas internas; e (iv) capacidade social: a possibilidade de interagir com outros agentes (WOOLDRIDGE; JENNINGS, 1995). Com o objetivo de melhorar seus comportamentos e decisões, os agentes são capazes de incorporar técnicas de inteligência artificial (por exemplo, técnicas de aprendizagem, buscas e sistemas especialistas).

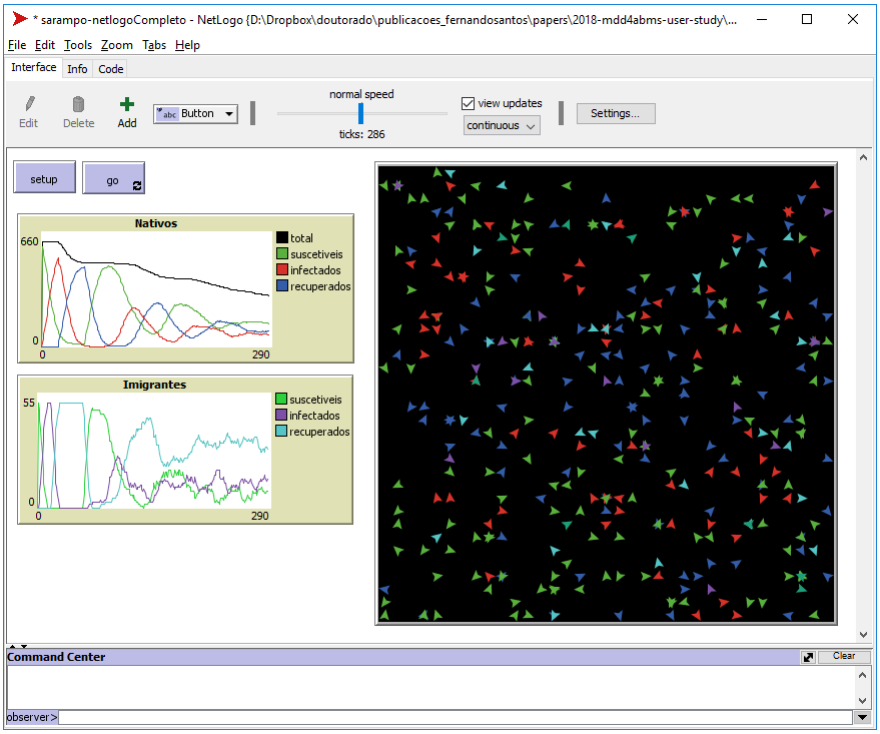
Simulação com agentes é um paradigma de simulação que usa agentes simulados para reproduzir, entender e predizer fenômenos (KLÜGL; BAZZAN, 2012). Um dos benefícios de utilizar essa simulação é a possibilidade de analisar e observar o fenômeno em estudo em dois níveis: em nível do agente e em nível macroscópico. No nível do agente é possível estudar diferentes estratégias de tomada de decisão dos agentes. Pode-se também considerar de forma natural a heterogeneidade dos agentes, já que este paradigma de simulação foca nos indivíduos. Já no nível macro pode-se estudar os resultados gerados a partir das ações e interações dos agentes entre si e com o ambiente (KLÜGL, 2008).

Segundo Klügl e Bazzan (2012), para desenvolver uma simulação com agentes é necessário especificar três elementos: um conjunto de agentes autônomos; as interações entre os agentes e ambiente, responsáveis por produzir a saída geral do sistema; e o ambiente simulado, que contém todos os demais elementos de simulação, como recursos e outros objetos sem comportamento ativo.

Um exemplo de simulação com agentes é apresentado na figura 1. O propósito desta simulação é simular a propagação de doenças com dois tipos de agentes: nativos e imigrantes. A doença é propagada a partir das interações entre os agentes, neste caso por contato ou proximi-

dade. O tipo do agente determina os parâmetros de transmissão e recuperação da doença, sendo que os nativos são mais suscetíveis a contrair e propagar a doença. No exemplo, o ambiente é tratado como uma grade por onde os agentes podem se movimentar. A saída da simulação (gráficos e visualização dos agentes) possibilita observar variações ao longo do tempo na população de agentes suscetíveis, infectados, e recuperados da doença.

Figura 1 – Exemplo de simulação com agentes: propagação de doenças



Fonte: Santos (2019).

2.2 PLATAFORMA DE SIMULAÇÃO NETLOGO

NetLogo é uma linguagem de programação multiagente e ambiente de modelagem para simular fenômenos naturais e sociais complexos. É particularmente adequada para modelar sistemas complexos que evoluem ao longo do tempo. Os desenvolvedores podem dar instruções para centenas ou milhares de agentes independentes operando simultaneamente, a fim de explorar as conexões entre os comportamentos de nível micro dos indivíduos e os padrões de nível macro que emergem de suas interações (TISUE; WILENSKY, 2004).

O NetLogo permite aos usuários abrir simulações e "brincar" com elas, explorando seu comportamento sob várias condições. O NetLogo é também um ambiente de autoria que é simples o suficiente para permitir que estudantes e pesquisadores criem seus próprios modelos, mesmo que não sejam programadores profissionais (TISUE; WILENSKY, 2004).

O NetLogo disponibiliza uma linguagem de programação completa, onde os desenvolvedores podem escrever procedimentos e usá-los como comandos embutidos. A partir da versão 2.0.1 do NetLogo, uma API passou a ser oferecida para extensões, para que desse modo os desenvolvedores possam adicionar novos elementos à linguagem, implementando-os diretamente em Java. Isso permite que os mesmos adicionem novos tipos de recursos ao NetLogo (TISUE; WILENSKY, 2004).

Como já dito anteriormente, os agentes podem incorporar técnicas de aprendizagem. Porém, quando em uma simulação o desenvolvedor necessita incorporar a um agente o comportamento de aprendizagem por reforço, ele mesmo deve codificar tal comportamento, já que extensões de aprendizado por reforço não existem para o NetLogo. Para que os desenvolvedores ganhem produtividade se faz necessário a implementação de uma extensão que permita a adoção simples do comportamento de aprendizagem por reforço a um agente.

Na linguagem de programação do NetLogo é possível definir raças de agentes. Isto é feito utilizando o comando `breed`, que pode ser visto na figura 2 linha 1. O comando `breed` espera um forma de identificar a raça no plural e uma forma de identificá-la no singular, por isso passamos `Cachorros` e `Cachorro`. É possível também definir atributos para os indivíduos das raças. Isto pode ser observado na linha 2 da figura 2 onde é especificado que toda indivíduo da raça `Cachorro` possui um atributo `latido`.

Existe uma convenção entre os desenvolvedores de simulações com agentes que utilizam o NetLogo onde deve-se criar dois procedimentos: `setup` para inicializar a simulação, e `go` para executar cada passo da simulação. O procedimento `setup` é mostrado na figura 2 linhas 4 à 6. Neste procedimento está sendo criado 10 agentes da raça `cachorro` e definido qual deve ser o latido desses agentes. Já o procedimento `go` é mostrado na figura 2 linhas 8 à 13.

Outro comando NetLogo relevante para este trabalho é o `ask`. Este comando é utilizado para dar instruções aos agentes. Todo código que deve ser executado por um agente deve estar localizado dentro de um contexto de agente. Uma das maneiras de estabelecer um contexto de agente é utilizando o comando `ask`. Um `ask` pode ser feito para todos os agentes ou apenas para alguns. Na linha 9 da figura 2 pode-se ver a utilização do comando `ask`. Nesse trecho do código pedimos para cada `cachorro` andar uma posição para frente (figura 2 linha 10) e exibir no terminal seu latido (figura 2 linha 11).

2.3 APRENDIZAGEM POR REFORÇO

A ideia de que aprendemos interagindo com nosso ambiente é provavelmente a primeira a ocorrer quando pensamos sobre a natureza da aprendizagem. Quando uma criança brinca, agita os braços ou olha em volta, ela não tem um professor explícito, mas tem uma conexão

Figura 2 – Exemplo de código NetLogo

```
1  breed [ Cachorros Cachorro ]
2  Cachorros-own [latido]
3
4  to setup
5    create-Cachorros 10 [set latido "auau"]
6  end
7
8  to go
9    ask Cachorros [
10     forward 1
11     print latido
12   ]
13 end
```

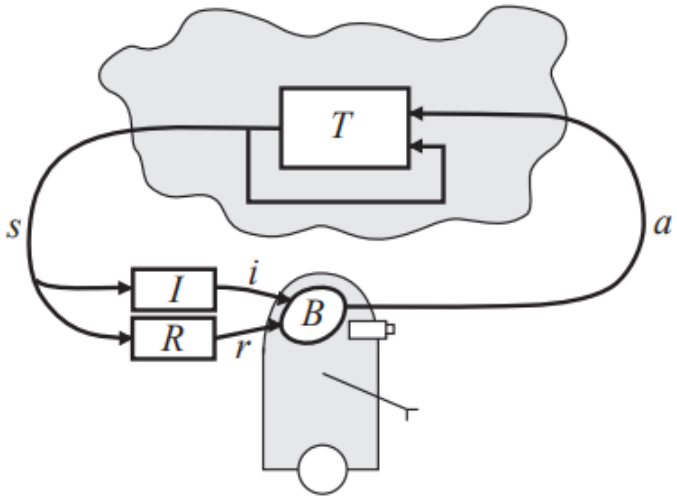
Fonte: O autor.

sensório-motora direta com seu ambiente. O exercício dessa conexão produz uma grande quantidade de informações sobre causa e efeito, sobre as consequências das ações e sobre o que fazer para atingir objetivos. Ao longo de nossas vidas, essas interações são, sem dúvida, uma importante fonte de conhecimento sobre nosso ambiente e sobre nós mesmos. Quer estejamos aprendendo a conduzir um carro ou a conversar, estamos conscientes de como o nosso ambiente responde ao que fazemos e procuramos influenciar o que acontece através do nosso comportamento. Aprender com a interação é uma ideia fundamental subjacente a quase todas as teorias da aprendizagem e da inteligência (SUTTON; BARTO, 2018).

No modelo padrão de aprendizagem por reforço apresentado na figura 3, um agente é conectado ao seu ambiente via percepção e ação. Em cada etapa de interação, o agente recebe como entrada i alguma indicação do estado atual s do ambiente; o agente então escolhe uma ação a para gerar como saída. A ação altera o estado do ambiente e o valor dessa transição de estado é comunicado ao agente por meio de um sinal de reforço escalar, r. O comportamento B do agente deve escolher ações que tendem a aumentar a soma dos valores de sinal de reforço a longo prazo. Ele pode aprender a fazer isso ao longo do tempo por tentativa e erro sistemáticos, guiados por uma ampla variedade de algoritmos (KAELBLING; LITTMAN; MOORE, 1996).

Para completar a definição do ambiente, é necessário especificar uma função de utilidade para o agente. Como o problema é de decisão sequencial, a função de utilidade dependerá de uma sequência de estados em vez de depender de um único estado. Em cada estado s pertencente ao conjunto de estados S do problema, o agente recebe uma recompensa $R(s)$, que pode ser positiva ou negativa, mas deve ser limitada (RUSSEL; NORVIG, 2004). A utilidade do agente é definida em termos da utilidade de sequências de estados. Em termos aproximados, a utilidade de um estado é a utilidade esperada das sequências de estados que poderiam segui-

Figura 3 – Modelo padrão de aprendizagem por reforço



Fonte: Kaelbling, Littman e Moore (1996).

lo. Ela é determinada considerando a recompensa imediata correspondente ao estado atual, somada à utilidade descontada esperada do próximo estado, supondo-se que o agente escolha a ação ótima. Dada essa definição, a utilidade verdadeira de um estado é denotada por $U(s)$. É importante notar 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).

Um dos métodos de aprendizagem por reforço é a aprendizagem de diferença temporal (DT), ou *temporal difference*. A ideia básica de todos os métodos de diferença temporal é definir primeiro as condições que são válidas no estado s quando as estimativas de utilidade estão corretas e, em seguida, escrever uma equação de atualização que move as estimativas em direção a uma equação de "equilíbrio" ideal (RUSSEL; NORVIG, 2004).

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

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

A equação de equilíbrio ideal é uma equação de restrição que deve se manter em equilíbrio quando os valores de utilidade estiverem corretos. Essa equação é importante pois permite determinar se o agente obteve uma politica ótima ou não, sendo que uma política especifica o

que o agente deve fazer para qualquer estado que possa alcançar. 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).

No caso do *Q-learning*, o equilíbrio ideal é dado pela equação 2.2. Esta equação deve se manter em equilíbrio quando os valores de Q estiverem corretos. Nela, γ é um fator de desconto que descreve a preferência do agente por recompensas atuais sobre recompensas futuras. $T(s,a,s')$ é a probabilidade de alcançar o estado s' ao executar a ação a no estado s. Por fim, a' é ação com maior valor no estado s'.

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

A equação 2.3, que é utilizada sempre que ocorre uma mudança de estado, faz de fato o agente alcançar o equilíbrio dado pela equação 2.1. Nela, α é um parâmetro que representa a taxa de aprendizagem.

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

O projeto de agente completo para um agente exploratório de aprendizagem Q usando DT é mostrado na figura 4. O agente recebe como entrada um conjunto de estados S, um conjunto de ações A, uma taxa de aprendizagem α e um fator de desconto γ. O agente então executa uma quantidade previamente estipulada de episódios. Em cada episódio o agente parte de um estado inicial — isto é, a configuração inicial do ambiente — e passa por um número indeterminado de estados até alcançar um estado terminal — ou seja, uma configuração do ambiente que caracteriza o fim de um episódio (por exemplo, quando o agente cumpriu ou falhou em cumprir seu objetivo). Cada episódio começa com a seleção de um estado inicial. Após isso o agente escolhe uma ação para o estado atual, executa ação, observa o novo estado e a recompensa recebida para então atualizar a tabela Q. Este processo se repete até o agente atingir um estado terminal.

Para selecionar uma ação, um agente precisa de um método de seleção de ação. Os métodos de seleção de ação gulosos sempre exploram o conhecimento atual para maximizar a recompensa imediata; eles não passam tempo nenhum testando ações aparentemente inferiores para ver se elas podem ser melhores. Uma alternativa é se comportar de forma gulosa a maior parte do tempo, mas, de vez enquanto, com uma pequena probabilidade ε, selecionar uma ação aleatória do conjunto de ações, independentemente do valor estimado de ação-valor. Esses métodos de seleção de ação que utilizam essa regra quase gulosa são chamados de $\varepsilon - greedy$. Uma vantagem desses métodos é que no limite, a medita que o número de passos aumenta, toda ação será amostrada um número infinito de vezes, portanto assegurando que o valor estimado

Figura 4 – Agente exploratório de aprendizagem Q usando DT

Algoritmo 1: AGENTE EXPLORATÓRIO DE APRENDIZAGEM Q USANDO DT

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

Fonte: Russel e Norvig (2004).

de exercer uma determinada ação converge para o valor real de exercer essa mesma ação. Isso, obviamente, implica que a probabilidade de selecionar uma ação ótima converge para valores maiores que $1 - \varepsilon$

2.4 TRABALHOS CORRELATOS

Nesta seção são apresentados trabalhos que de algum modo se relacionam com o trabalho que está sendo proposto. No primeiro trabalho desenvolveu-se um agente para controle semafórico que aprende por meio do método *Q-learning* e é testado através de simulações com agentes. No segundo é desenvolvido uma biblioteca de aprendizagem por reforço onde um dos algoritmos suportados é o *Q-learning*. Já no terceiro é desenvolvido uma extensão para a plataforma NetLogo que busca facilitar o uso de lógica difusa em simulações com agentes.

2.4.1 *An Agent-Based Learning Towards Decentralized and Coordinated Traffic Signal Control*

El-Tantawy e Abdulhai (2010) desenvolveram um agente para controle semafórico que usa o *Q-learning* para aprender quais pistas liberar de modo a minimizar o atraso de cada veículo no semáforo. Três modelos diferentes do problema foram avaliados para determinar o mais apropriada para o problema. Cada modelo adota diferentes representações de estado, porém, as ações e recompensas são as mesmas em todos os diferentes modelos.

Com relação às representações de estado, todas adotam um vetor de tamanho N, onde N é o número de fases possíveis para o semáforo (direções em que o fluxo de veículos é permitido). A diferença entre as representações é que cada uma especifica um componente diferente para ocupar as posições do vetor. Os possíveis componentes do vetor são: i) o tamanho da maior fila daquela fase se a mesma está fechada e o número de carros que passou pela fase se ela está aberta; ii) o tamanho da maior fila daquela fase; e iii) o somatório do atraso de cada veículo que esta parado naquela fase. As ações possíveis são trocar de fase e a recompensa é a mudança no somatório do atraso de todos os veículos.

Todos os modelos desenvolvidos foram testados com volumes de tráfego estáticos e dinâmicos ao longo do tempo. As simulações consideraram que o agente está situado em um cruzamento importante no centro da cidade de Toronto, no Canadá. A quantidade de veículos que transita em cada pista foi definida a partir de dados reais. O treinamento do agente ocorreu no ambiente de simulações de tráfego Paramics.

2.4.2 *Developing a Python Reinforcement Learning Library for Traffic Simulation*

No trabalho de Ramos, Lemos e Bazzan (2017) é desenvolvido uma biblioteca de aprendizagem por reforço para a linguagem de programação Python chamada PyRL. Um dos algoritmos suportados pela biblioteca PyRL é justamente o *Q-learning*. A principal vantagem de utilizar a PyRL é sua conectividade com a plataforma de simulações de tráfego SUMO[1].

Para utilizar a PyRL o autor fornece métodos abstratos que devem ser implementados de acordo com o problema em questão. Para modelar o ambiente são definidos métodos abstratos para iniciar o mesmo, para obter as ações em um estado, para executar um episódio e para executar um passo de um episódio. Para definir o agente são utilizados métodos abstratos que permitem que o mesmo seja iniciado, que aja e que receba uma recompensa.

2.4.3 *Fuzzy Logic for Social Simulation Using NetLogo*

Izquierdo et al. (2015) apresentam uma extensão NetLogo para facilitar o uso de lógica difusa em simulações com agentes. A partir das funções fornecidas pela extensão é possível modelar agentes que são capazes de seguir regras que incluem termos imprecisos. Para facilitar o entendimento das funções fornecidas na extensão, Izquierdo et al. (2015) apresentam um tutorial passo a passo de como modelar uma simulação onde os agentes utilizam lógica difusa. As funcionalidades da extensão foram inicialmente implementadas através de uma biblioteca de funções escritas diretamente na linguagem de programação da plataforma NetLogo, e posteri-

[1] https://sumo.dlr.de/index.html

ormente foram convertidas para uma extensão implementada em Java. A vantagem da extensão em relação a biblioteca é que ela é computacionalmente mais eficiente e mais fácil de usar. O código desenvolvido é aberto, o que possibilita sua inspeção.

2.4.4 Considerações Sobre os Trabalhos Correlatos

Esse trabalho, assim como o trabalho de Izquierdo et al. (2015), propõe o desenvolvimento de uma extensão para a plataforma NetLogo. A diferença está em que no trabalho de Izquierdo et al. (2015) foi desenvolvido uma extensão para utilização de lógica difusa no NetLogo, enquanto a extensão que será desenvolvida nesse trabalho buscará atender a necessidade da utilização do algoritmo *Q-Learning* na plataforma NetLogo.

Uma caso semelhante ao proposto nesse trabalho é o de Ramos, Lemos e Bazzan (2017). Neste trabalho, os autores desenvolveram uma biblioteca de aprendizagem por reforço para Python chamada PyRL. Entretanto, a PyRL não se conecta com a plataforma de simulações NetLogo, por isso a necessidade de uma extensão que faça isso. O trabalho de El-Tantawy e Abdulhai (2010) demonstra um caso de uso real do *Q-Learning*, o qual será levado em consideração no desenvolvimento da extensão.

3 EXTENSÃO NETLOGO PARA *Q-LEARNING*

As extensões para a plataforma NetLogo possuem o intuito de fornecer novos comandos primitivos para a plataforma, visando facilitar a vida do desenvolvedor de simulações com agentes. Portanto, este trabalho se propôs a desenvolver uma extensão que forneça comandos primitivos para incorporar, nos agentes da simulação, o comportamento de aprendizagem por reforço através do método *Q-learning*. A extensão foi desenvolvida com a linguagem de programação Scala e utilizou a API para desenvolvimento de extensões que passou a ser oferecida a partir da versão 2.0.1 da plataforma NetLogo.

A extensão disponibiliza meios para que o desenvolvedor especifique: i) os estados do problema de aprendizagem; ii) as ações que o agente pode executar; iii) os valores de recompensa em cada estado; iv) o método de seleção de ação a ser utilizado; v) o valor de fator de desconto γ e de taxa de aprendizagem α; vi) uma maneira de fazer o agente aprender; e vii) maneiras para depurar. As próximas subseções descrevem como cada um destes elementos é disponibilizado pela extensão desenvolvida.

Vale ressaltar que todos os comandos informados devem ser envoltos em blocos de código `ask`. Esses blocos de código são utilizados para instruir os agentes a fazer operações: se um bloco desse tipo é chamado para uma determinada `breed`, ou raça, do NetLogo, todos os agentes pertencentes aquela raça irão executar aquele determinado bloco de código. Um `ask` pode ser executado para um ou mais agentes específicos, não apenas para uma raça inteira.

3.1 ESPECIFICAÇÃO DOS ESTADOS

Para poder funcionar de maneira apropriada a extensão precisa de uma forma de caracterizar cada possível estado. A extensão desenvolvida permite aos desenvolvedores de simulações com agentes caracterizar a representação do estado de duas maneiras. A primeira é especificando características do agente aprendiz que devem ser levadas em conta na hora de criar uma representação de um estado. Já a segunda, que busca complementar a primeira, permite especificar uma função que retorna um texto que ajuda a caracterizar um estado, dessa maneira o desenvolvedor possui flexibilidade para adicionar, na representação do estado, características que não são do agente aprendiz. Um exemplo de uso para a segunda maneira seria caso alguma variável externa ao agente fosse útil na representação do estado, como por exemplo: velocidade do vento no ambiente, quantidade de determinados objetos no ambiente e o tempo em que aquele estado é visitado.

Levando em conta a necessidade dessas duas maneiras, a extensão desenvolvida disponibiliza dois novos comandos primitivos ao NetLogo. A figura 5 é uma implementação parcial de uma simulação que utiliza a extensão e será utilizada para exemplificar o uso dos comandos fornecidos pela extensão. No procedimento `setup` implementado nas linhas 1 a 12 é onde se realiza a configuração da simulação. Entre as linhas 3 e 11 pode-se verificar um bloco `ask`, que solicita que todos os agentes da raça `Walkers` executem os comandos de configuração da aprendizagem por reforço Q. Para especificar os estados de acordo com a primeira maneira o desenvolvedor deve utilizar o comando primitivo apresentado na linha 4. Este comando recebe como argumento uma lista de nomes de variáveis (atributos) que o agente possui. Já para especificar os estados de acordo com a segunda maneira o comando primitivo utilizado deve ser o apresentado na figura 6 linha 5. Este, além de receber como argumento uma lista de nomes de variáveis que o agente possui, recebe também o nome de *reporter*. Em NetLogo, um *reporter* é um procedimento que retorna um valor. Esse valor deve ser um texto, que será adicionado na representação dos estados.

Os atributos e a função passada para a extensão na hora de especificar o estados são utilizados posteriormente pela extensão toda vez que o agente visita um novo estado para criar uma caracterização desse estado em tempo de execução. Dessa maneira é possível saber se é um estado nunca antes vistado ou não e também localizar a posição correta na tabela Q que deve ser atualizada.

Um agente deve utilizar um dos dois comandos primitivos para receber sua caracterização de estado, nunca ambas. A utilização desse primitivo por um agente implica que ele é agora um agente aprendiz. Os comandos que serão citados ao decorrer desse capítulo devem ser executados apenas se os estado já estiverem sido especificados. Caso contrário, a extensão lançará uma exceção pois não reconhece o agente como aprendiz.

3.2 ESPECIFICAÇÃO DAS AÇÕES

As ações que o agente aprendiz pode executar devem ser especificadas como procedimentos NetLogo. Portanto, o desenvolvedor deve implementar procedimentos para cada ação e depois informar à extensão quais são esses procedimentos. Para fazer isso ele deve utilizar o comando primitivo mostrado na figura 5, linha 5. Este comando recebe como argumento procedimentos NetLogo entre colchetes. Cada procedimento passado como argumento será armazenado pela extensão e executado quando for oportuno. Não existe limite de ações, o desenvolvedor pode criar quantas forem necessárias, porém, para que isso seja possível, ele deve manter a chamada do comando entre parênteses. Na linha 20 da figura 5 é mostrada a implementação de um procedimento que implementa a ação `goUp` do agente.

Figura 5 – Exemplo de implementação de uma simulação utilizando a extensão

```
to setup
    clear-all
    ask Walkers [
        qlearningextension:state-def ["xcor" "ycor"]
        (qlearningextension:actions [goUp] [goDown] [goLeft] [goRight])
        qlearningextension:end-episode  [isEndState] resetEpisode
        qlearningextension:reward [rewardFunc]
        qlearningextension:action-selection "random-normal" [0]
        qlearningextension:learning-rate 1
        qlearningextension:discount-factor 1
    ]
end

to go
    ask Walkers [
        qlearningextension:learning
    ]
end

to goUp
  if ycor + 1 != max-pycor + 1 [
    set heading 0
    fd 1
  ]
end

to-report rewardFunc
  report [reward] of patch-here
end

to-report isEndState
  if [pcolor] of patch-here = blue or [pcolor] of patch-here = green [
    report true
  ]
  report false
end

to resetEpisode
  setxy -4 -4
end
```

Fonte: O autor.

3.3 ESPECIFICAÇÃO DAS RECOMPENSAS

Para especificar a recompensa, o desenvolvedor deve apenas informar qual é o *reporter* que implementa o cálculo da recompensa. Este *reporter* deve retornar um valor numérico que

Figura 6 – Comando primitivo fornecido pela extensão: `state-def-extra`

```
1  globals[velocidadeDoVento]
2
3  to setup
4       ask Walkers [
5            qlearningextension:state-def-extra ["xcor" "ycor"] [stateAux]
6       ]
7  end
8
9  to-report stateAux
10   report velocidadeDoVento
11 end
```

Fonte: O autor.

representa a recompensa que o agente recebe ao estar naquele estado. Para informar à extensão qual *reporter* será responsável por retornar a recompensa dos estado o desenvolvedor deve utilizar o comando primitivo mostrado na figura 5 linha 7. Este comando recebe como argumento justamente o *reporter* entre colchetes. O *reporter* passado no comando supracitado tem sua implementação exibida na figura 5 linha 27.

3.4 ESPECIFICAÇÃO DE EPISÓDIO

Para saber se um estado representa o fim de um episódio o desenvolvedor deve implementar um *reporter* que verifica o estado atual e através de um retorno booleano informa se esse é ou não um estado final. Além disso, o desenvolvedor deve implementar um procedimento que será executado sempre que um episódio acabar. Uma das responsabilidades deste procedimento deve ser reiniciar o ambiente e/ou o agente.

O comando primitivo que permite ao desenvolvedor especificar o *reporter* que identifica um estado final e o procedimento que reinicia o ambiente é mostrado na figura 5 linha 6. Este comando requer dois argumentos. O primeiro argumento é o *reporter* que verifica se chegou ao fim de um episódio. Ele deve ser informado entre colchetes para que a extensão possa reconhecê-lo como um *reporter*. O segundo argumento é o procedimento que reinicia o ambiente/agente. Estes procedimentos passados como argumento são mostrados na figura 5 nas linhas 31 e 38.

3.5 ESPECIFICAÇÃO DO MÉTODO DE SELEÇÃO DE AÇÃO

A extensão disponibiliza duas estratégias para seleção de ação. A primeira é a melhor ação ou uma ação aleatória de acordo com uma taxa passado pelo desenvolvedor. A segunda é

a estratégia *e-greedy*, detalhada na seção 2.3. Um agente deve possuir apenas uma estratégia de seleção de ação.

Para especificar o método de seleção de ação o comando usado é o `action-selection`. Ele recebe como argumento o nome da estratégia e uma lista contendo valores utilizados pelas estratégias. A figura 5, linha 8, mostra como utilizar o comando para usar a primeira estratégia, chamada de *random-normal*. Essa estratégia recebe, entre colchetes, a taxa que determina a porcentagem de ações aleatórias que serão tomadas. A figura 7 mostra como utilizar o comando primitivo para usar a segunda estratégia, a *e-greedy*. Essa estratégia recebe na lista o valor do ε e valor do decaimento de ε.

Figura 7 – Comando primitivo fornecido pela extensão: `action-selecion`

```
1 qlearningextension:action-selection "e-greedy" [0.7 0.99995]
```

Fonte: O autor.

3.6 APRENDIZAGEM

Para especificar o parâmetro de taxa de aprendizagem α a extensão disponibiliza o comando mostrado na figura 5 linha 9. Já para especificar o fator de desconto γ a extensão disponibiliza o comando mostrado na 5 linha 10.

A aprendizagem ocorre por meio do comando primitivo apresentado na figura 5 linha 16. Esse comando é o que proporciona o maior ganho para o desenvolvedor, pois ele encapsula todo o algoritmo de aprendizado, realizando todas as seguintes ações: seleciona uma ação de acordo com o método escolhido, executa a ação, recebe a recompensa de estar no novo estado, atualiza a tabela Q, verifica se o novo estado é um estado terminal e se for executa o procedimento de reinício de ambiente.

Caso o usuário deseje obter o episódio atual para realizar alguma lógica ele pode executar o comando primitivo apresentado na figura 8 em conjunto com a declaração `let` para declaração de variável com o fim de obter o mesmo. Esse comando retorna um número inteiro que identifica o episódio atual.

Figura 8 – Comando primitivo fornecido pela extensão: `episode`

```
1 let episode qlearningextension:episode
```

Fonte: O autor.

3.7 DEPURAÇÃO

A extensão possui dois meios para auxiliar na depuração. O primeiro consiste em passar um argumento booleano como verdadeiro para a função `qlearningextension:learning`. Esse argumento faz com que a cada execução da aprendizagem as seguintes informações sejam exibidas no terminal: o estado anterior; a lista Q anterior (ou seja, os valores da tabela Q para o estado anterior); a recompensa do novo estado; o novo estado; o maior valor Q do novo estado e a nova lista Q (ou seja, os valores da tabela Q para o novo estado). A chamada do comando `learning` para que isso seja possível é mostrada na figura 9. Outra maneira de depurar é utilizando um comando primitivo que retorna a tabela Q, esse comando é mostrado na figura 10.

Figura 9 – Comando primitivo fornecido pela extensão: `learning true`

```
1 (qlearningextension:learning true)
```

Fonte: O autor.

Figura 10 – Comando primitivo fornecido pela extensão: `get-qtable`

```
1 print(qlearningextension:get−qtable)
```

Fonte: O autor.

3.8 DISPONIBILIZAÇÃO DA EXTENSÃO PARA USO DA COMUNIDADE

A extensão desenvolvida foi submetida a comunidade através do NetLogo *Extension Manager*, uma ferramente que permite descobrir e gerenciar extensões diretamente da plataforma. Para adicionar uma extensão ao *Extension Manager* é preciso adicionar um *Pull Request* a um repositório Git gerenciado pela equipe do NetLogo.[1] Esta equipe avalia o pedido e torna a extensão disponível por meio do *Extension Manager*.

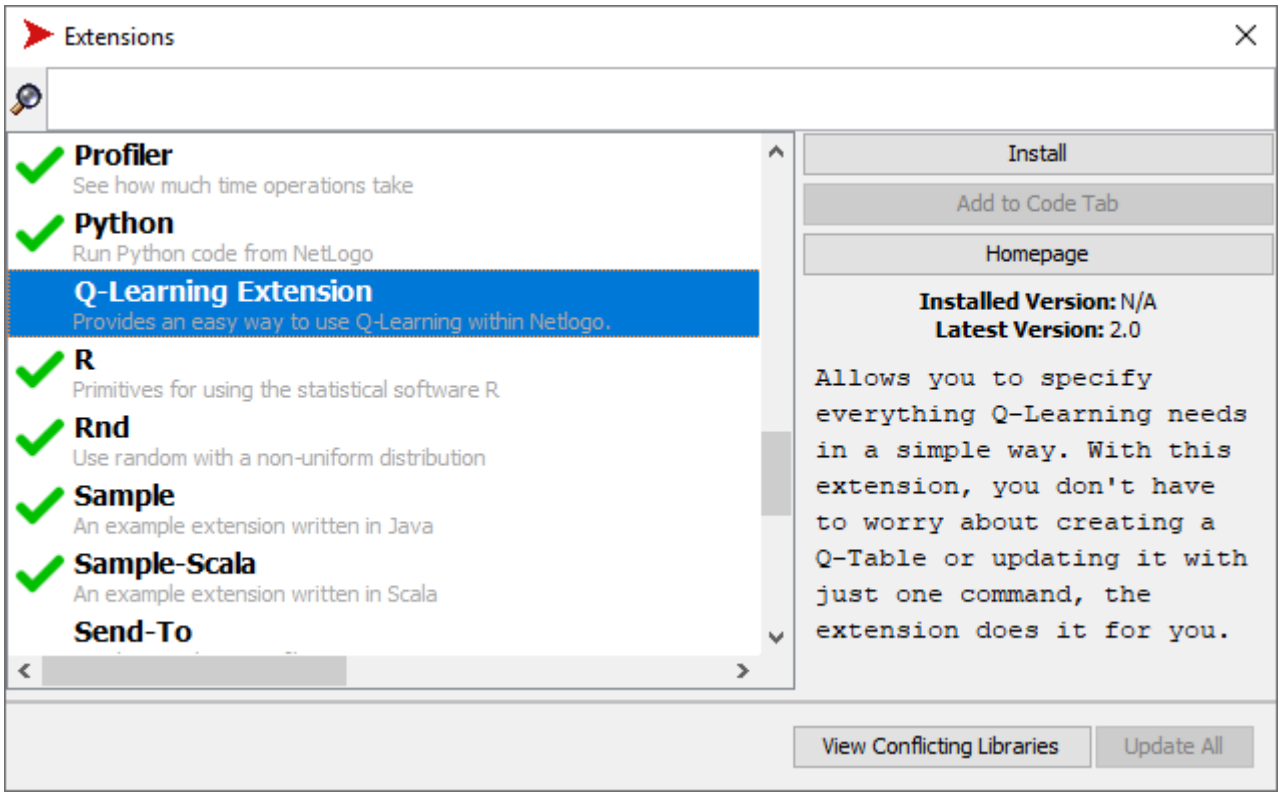
O *Pull Request* realizado para disponibilizar a extensão foi feito no dia 28 de outubro de 2019 e aceito no dia 31 de outubro de 2019. Houve uma sugestão para que as chamadas dos comandos da extensão começassem apenas com `qlearning:` ou `ql:` em vez do atual `qlearningextension:`. A justificativa para a sugestão foi que a palavra *extension* não acrescenta muito no código quando se trabalha com NetLogo. A sugestão será acatada numa nova versão da extensão.

A partir da inclusão da extensão no *Extension Manager*, ela está a quatro cliques de distância de qualquer usuário do NetLogo. Apenas é necessário que ele acesse a opção de *extensions*

[1] https://github.com/NetLogo/NetLogo-Libraries#netlogo-libraries

através do menu de ferramentas do NetLogo, procure pela extensão e faça sua instalação. A tela do *Extension Manager* com a extensão desenvolvida é apresentada na figura 11.

Figura 11 – A extensão desenvolvida disponibilizada no *Extension Manager* do NetLogo



Fonte: O Autor.

Além da inclusão da extensão no *Extension Manager*, foi desenvolvida uma documentação de uso da extensão. A documentação está disponível online[2], e apresenta a explicação de como utilizar cada comando primitivo que a extensão oferece, além de mostrar a implementação de um exemplo prático de uso da extensão.

[2]https://github.com/KevinKons/qlearning-netlogo-extension

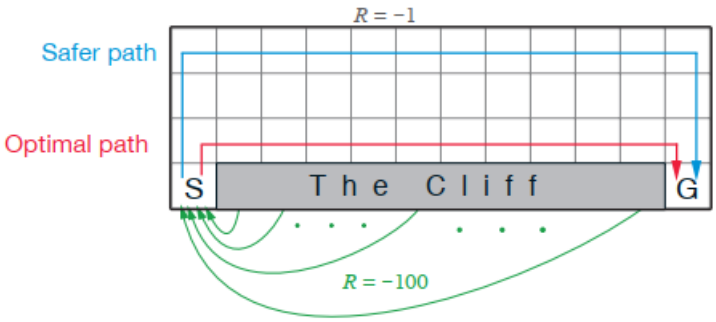
4 AVALIAÇÃO DA EXTENSÃO

A extensão desenvolvida foi avaliada de duas maneiras, a primeira busca validar a corretude da implementação do algoritmo *Q-Learning* e a segunda avaliar as características do código de uma simulação com agentes que utilize a extensão.

4.1 VALIDAÇÃO DA IMPLEMENTAÇÃO

Para validar a implementação do *Q-Learning* uma simulação de um cenário clássico de aprendizagem por reforço foi desenvolvido utilizando a extensão. O cenário escolhido para ser implementado foi o *Cliff Walking*. Neste cenário o ambiente é um mundo bi-dimensional, mostrado na figura 12, com tarefas episódicas, estado inicial e objetivo, e as ações de ir para cima, para baixo, para a direita e para a esquerda. Nesse cenário, o estado inicial está marcado com a letra S na figura 12 e o estado objetivo com a letra G na figura 12. A Recompensa é -1 em todas as transições, com exceção daquelas para a região marcada como "*The Cliff*". Ir para essa região implica em uma recompensa de -100 e envia o agente de volta para o estado inicial (SUTTON; BARTO, 2018). A linha vermelha na figura 12 denota o caminho ótimo a ser percorrido pelo agente para chegar em seu objetivo.

Figura 12 – Cenário *Cliff Walking*



Fonte: Sutton e Barto (2018).

Para validação, foram feitas 10 execuções do aprendizado no cenário *Cliff Walking*. A cada execução, o agente repetia 100 episódios de aprendizagem. A partir das tabelas Q das 10 execuções, foram calculadas a média e desvio padrões dos valores Q para cada par de estado-ação. A partir destas médias dos pares de estado-ação foi examinada a política que o algoritmo escolheu para verificar se ela está de acordo com a política ótima do cenário. A política ótima é a denotada em vermelho na figura 12.

Na simulação a definição de estado adotada foi a posição do agente no cenário, o método de seleção de ação foi o $\varepsilon - greedy$ onde o valor de ε adotado foi 0.4 e a taxa de diminuição

do valor de ε adotada foi 0.95. Já os parâmetros taxa de aprendizagem e fator de desconto foram ambos definidos como 1 por fornecerem um aprendizado mais rápido. Ao final dos 100 episódios o valor de ε restante foi 0.0022. A tabela 1 apresenta a média e desvio padrão dos pares de estado-ação para as 10 execuções do problema do *Cliff-Walking*. A primeira coluna da tabela representa um estado, as demais colunas representam a recompensa esperada de realizar aquela ação naquele estado.

Analisando a tabela 1 podemos perceber que a implementação do *Q-Learning* disponibilizada pela extensão desenvolvida nesse trabalho conseguiu encontrar a política ótima. Para observar isso basta olhar para os estados do caminho ótimo e ver se as maiores recompensas desses estado fazem o agente permanecer nesse caminho. Por exemplo, no estado (0,0) a política indica que o agente deve ir para cima, já nos estados (0,1) e (1,1) a política indica que o agente deve ir para direita. Na tabela 1 os valores destacados em negrito são aqueles que fazem o agente seguir a política ótima.

4.2 ANÁLISE DAS CARACTERÍSTICAS DO CÓDIGO DA SIMULAÇÃO

O objetivo da extensão desenvolvida é facilitar o uso do aprendizado *Q-learning* em simulações com agentes. Para verificar se a extensão traz este benefício ao desenvolvedor, foi realizada uma análise comparativa entre um código fonte de simulação implementado utilizando a extensão, e um código fonte implementado sem usar a extensão. A simulação escolhida é a *Maze*, onde um agente, uma formiga, deve andar pelo ambiente até encontrar o estado objetivo. No caminho ela não deve cair na água pois esse caracteriza um estado terminal e leva a formiga imediatamente de volta ao estado inicial. O ambiente é um mundo bi-dimensional onde as ações que o agente pode executar são ir para cima, para baixo, para a esquerda e para a direita. O agente não recebe nenhuma recompensa ao fazer movimentos, com exceção de quando ele se movimenta para a água, onde a recompensa é -10, e de quando se movimenta para o estado objetivo, onde a recompensa é 10. A figura 13 apresenta o cenário da simulação.

A simulação *Maze* foi escolhida para realizar esta análise pois o seu código fonte está disponível online, feita por um membro da comunidade NetLogo (ROOP, 2006). Deste modo, evita-se qualquer viés que a implementação sem a extensão pudesse ter nos resultados. A implementação disponível utiliza apenas comando nativos do NetLogo para realizar o desenvolvimento da simulação. A implementação original do modelo foi feita na versão 3 da plataforma NetLogo, e seu código foi traduzido para a versão 6 da plataforma para que fosse possível sua execução.

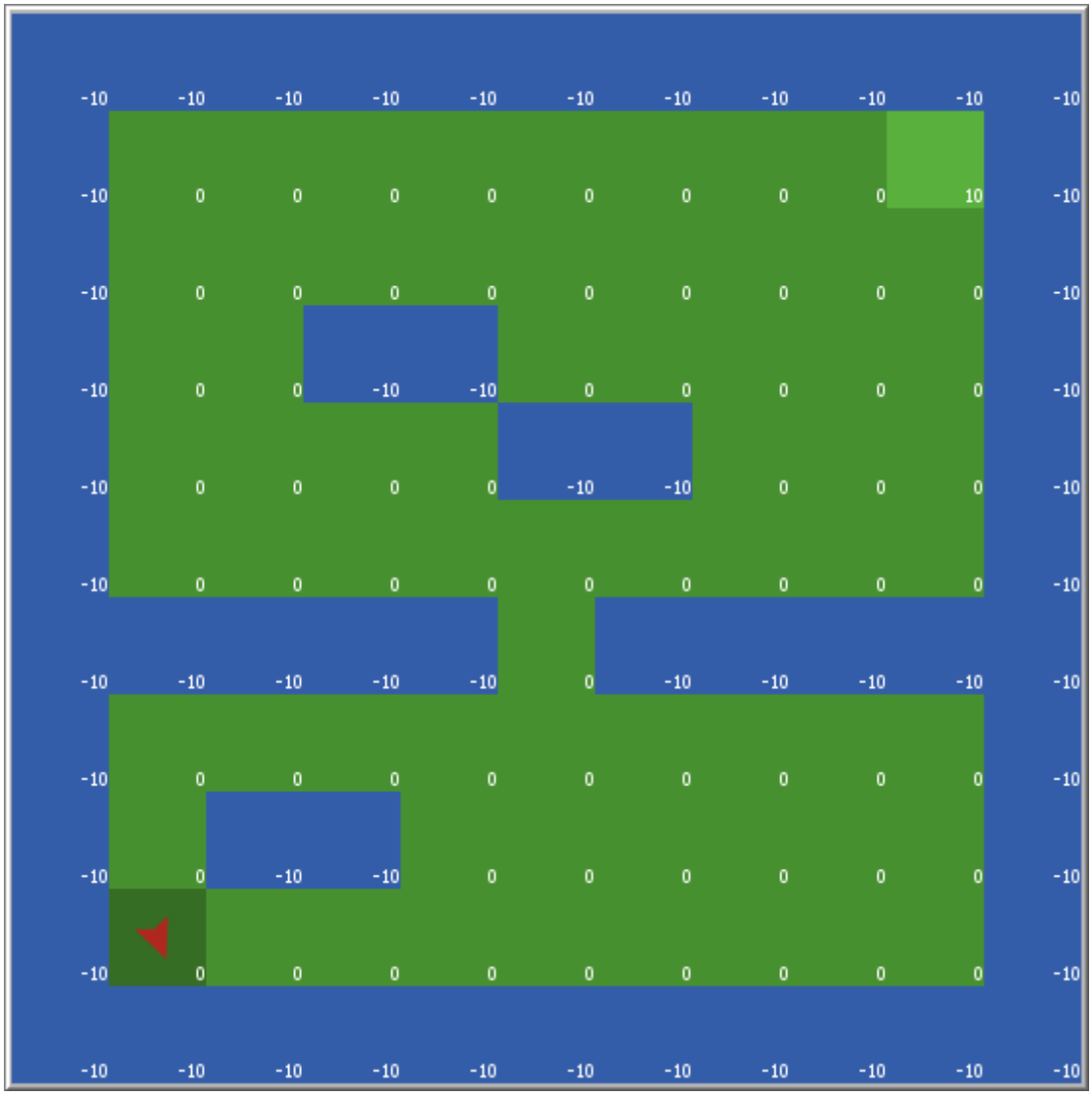
A figura 14 mostra a implementação do modelo sem o uso da extensão. Já a figura 15 mostra a implementação do modelo com o uso da extensão. Nas figuras 14 e 15 foram omitidas

as linhas de código referentes à configuração do ambiente. O modelo implementado utilizando a extensão possui mais linhas de código (52 linhas) que o modelo implementado sem o uso da

Tabela 1 – Média e desvio padrão das tabelas Q de 10 execuções do cenário *Cliff Walking*

	Valores Q			
Estado	Ação: Direita	Ação: Baixo	Ação: Esquerda	Ação: Cima
(0,0)	-100.0 ± 0.00	-14.0 ± 0.00	-14.0 ± 0.00	**-13.0 ± 0.00**
(0,1)	**-12.0 ± 0.00**	-13.4 ± 0.13	-13.0 ± 0.00	-13.0 ± 0.00
(0,2)	-12.10 ± 0.03	-12.9 ± 0.03	-12.0 ± 0.06	-12.2 ± 0.06
(0,3)	-11.4 ± 0.13	-12.3 ± 0.09	-11.0 ± 0.06	-11.6 ± 0.19
(0,4)	-11.8 ± 0.25	-11.0 ± 0.13	-11.0 ± 0.06	-11.8 ± 0.06
(1,1)	**-11.0 ± 0.00**	-100.0 ± 0.00	-12.0 ± 0.13	-12.1 ± 0.03
(1,2)	-11.4 ± 0.19	-11.8 ± 0.06	-12.3 ± 0.09	-11.5 ± 0.16
(1,3)	-11.1 ± 0.03	-11.5 ± 0.16	-11.8 ± 0.06	-11.4 ± 0.13
(1,4)	-11.0 ± 0.00	-11.6 ± 0.19	-11.4 ± 0.13	-11.5 ± 0.16
(2,1)	**-10.0 ± 0.00**	-100.0 ± 0.00	-11.4 ± 0.19	-11.3 ± 0.22
(2,2)	-10.9 ± 0.03	-10.9 ± 0.03	-11.7 ± 0.09	-11.1 ± 0.03
(2,3)	-10.7 ± 0.22	-11.2 ± 0.06	-11.3 ± 0.41	-11.0 ± 0.32
(2,4)	-10.5 ± 0.16	-10.7 ± 0.09	-11.1 ± 0.03	-10.9 ± 0.03
(3,1)	**-9.0 ± 0.00**	-100.0 ± 0.00	-10.5 ± 0.16	-10.1 ± 0.03
(3,2)	-9.9 ± 0.03	-10.0 ± 0.00	-10.8 ± 0.06	-10.5 ± 0.16
(3,3)	-10.0 ± 0.00	-10.6 ± 0.19	-10.5 ± 0.16	-10.3 ± 0.09
(3,4)	-9.7 ± 0.22	-10.3 ± 0.09	-10.5 ± 0.16	-10.2 ± 0.06
(4,1)	**-8.0 ± 0.00**	-100.0 ± 0.00	-9.7 ± 0.22	-9.3 ± 0.09
(4,2)	-9.0 ± 0.00	-9.0 ± 0.00	-9.8 ± 0.38	-9.4 ± 0.13
(4,3)	-9.2 ± 0.06	-9.5 ± 0.16	-10.1 ± 0.35	-9.6 ± 0.13
(4,4)	-9.3 ± 0.09	-9.7 ± 0.09	-9.9 ± 0.28	-9.5 ± 0.16
(5,1)	**-7.0 ± 0.00**	-100.0 ± 0.00	-8.7 ± 0.22	-8.3 ± 0.22
(5,2)	-8.0 ± 0.00	-8.0 ± 0.00	-9.2 ± 0.25	-8.8 ± 0.06
(5,3)	-8.3 ± 0.09	-8.7 ± 0.22	-9.2 ± 0.06	-8.8 ± 0.06
(5,4)	-8.5 ± 0.16	-8.9 ± 0.03	-9.2 ± 0.06	-8.5 ± 0.16
(6,1)	**-6.0 ± 0.00**	-100.0 ± 0.00	-7.3 ± 0.09	-7.2 ± 0.06
(6,2)	-7.0 ± 0.00	-7.0 ± 0.00	-8.1 ± 0.03	-7.9 ± 0.03
(6,3)	-7.5 ± 0.16	-7.8 ± 0.06	-8.3 ± 0.22	-8.1 ± 0.28
(6,4)	-7.8 ± 0.06	-8.0 ± 0.00	-8.6 ± 0.19	-8.4 ± 0.19
(7,1)	**-5.0 ± 0.00**	-100.0 ± 0.00	-6.5 ± 0.16	-6.3 ± 0.09
(7,2)	-6.0 ± 0.00	-6.0 ± 0.00	-6.9 ± 0.03	-7.0 ± 0.00
(7,3)	-6.8 ± 0.06	-7.0 ± 0.00	-7.7 ± 0.22	-7.5 ± 0.16
(7,4)	-7.1 ± 0.03	-7.3 ± 0.09	-7.9 ± 0.35	-7.6 ± 0.13
(8,1)	**-4.0 ± 0.00**	-100.0 ± 0.00	-5.7 ± 0.09	-5.6 ± 0.19
(8,2)	-5.0 ± 0.00	-5.0 ± 0.00	-6.3 ± 0.09	-6.1 ± 0.28
(8,3)	-5.9 ± 0.03	-6.0 ± 0.00	-6.7 ± 0.41	-6.9 ± 0.03
(8,4)	-6.1 ± 0.03	-6.6 ± 0.13	-7.1 ± 0.35	-6.7 ± 0.09
(9,1)	**-3.0 ± 0.00**	-100.0 ± 0.00	-4.4 ± 0.13	-4.6 ± 0.13
(9,2)	-4.0 ± 0.00	-4.0 ± 0.00	-5.3 ± 0.22	-5.5 ± 0.16
(9,3)	-5.0 ± 0.00	-5.0 ± 0.00	-5.5 ± 0.16	-5.7 ± 0.09
(9,4)	-5.5 ± 0.16	-5.7 ± 0.22	-6.0 ± 0.00	-5.8 ± 0.06
(10,1)	**-2.0 ± 0.00**	-100.0 ± 0.00	-3.7 ± 0.22	-3.8 ± 0.06
(10,2)	-3.0 ± 0.00	-3.0 ± 0.00	-4.2 ± 0.25	-4.3 ± 0.09
(10,3)	-4.0 ± 0.00	-4.0 ± 0.00	-5.0 ± 0.32	-4.7 ± 0.09
(10,4)	-5.0 ± 0.00	-4.9 ± 0.03	-5.7 ± 0.22	-5.3 ± 0.09
(11,1)	-2.0 ± 0.00	**-1.0 ± 0.00**	-2.7 ± 0.09	-2.5 ± 0.16
(11,2)	-3.0 ± 0.00	-2.0 ± 0.00	-3.5 ± 0.16	-3.4 ± 0.13
(11,3)	-4.0 ± 0.00	-3.0 ± 0.00	-4.1 ± 0.03	-4.4 ± 0.19
(11,4)	-4.8 ± 0.06	-4.0 ± 0.00	-5.0 ± 0.00	-4.8 ± 0.06

Figura 13 – Cenário *Maze*



Fonte: Roop (2006).

extensão (50 linhas). Apesar de pequena, a diferença pode levar a crer que o código utilizando a extensão é mais complexo. Porém, analisando os elementos abaixo descritos em cada código fonte, verifica-se que isto não acontece.

Especificação da tabela *Q*. A implementação da tabela *Q* no modelo original exige a definição de uma estrutura de dados para armazenar os valores *Q*. Cada posição, ou *patch*, que o agente pode ocupar guarda uma lista *Q*, isto é, os valores da tabela *Q* para aquele estado. Isto pode ser observado na linha 1 da figura 14. Já com o uso da extensão, a tabela *Q* é transparente ao desenvolvedor e ele não precisa definir qualquer estrutura de dados para ela. O desenvolvedor apenas precisa usar os comandos das linhas 11 e 12 da figura 15 para caracterizar os estados e ações possíveis e a partir disso é gerada a tabela *Q*.

Especificação das Ações. Para tornar a execução de ações possíveis o autor da implementação original necessitou criar uma lista de possíveis direções para as quais o agente pode andar (figura 14 linha 11). A execução da ação acontece nas linhas 43 e 49 da figura 14. Já para tornar a execução de ações possível com o uso da extensão foi necessário informar para a mesma quais são os procedimentos responsáveis pelas execuções das ações. A extensão recebe tais procedimentos na linha 12 da figura 15 e a implementação das ações acontece nas linhas 33 à 51.

Especificação das Recompensas. A recompensa no código que utiliza a extensão é definida também através de um *reporter*, informado à extensão na linha 14 da figura 15. Este *reporter* é implementado na linha 29 à 31. Já sem o uso da extensão, para obter a recompensa, foi necessário apenas a linha 44 da figura 14. Contudo, o uso de um *reporter* para obtenção da recompensa permite desacoplar a implementação de obtenção de recompensa do algoritmo de aprendizagem.

Especificação de Episódio. A cláusula de término de um episódio, no código original, foi definida na linha 24 da figura 14 e se deu pelo uso de um laço de repetição. Ainda no código original, o comando responsável por reiniciar um episódio está na linha 23 da figura 14, fora desse mesmo laço de repetição. Por outro lado, com o uso da extensão a cláusula de término e a *procedure* de reinício de episódio são passados para a extensão através de uma *procedure* e um *reporter* na linha 13 da figura 15. A implementação da *procedure* e do *reporter* se dão respectivamente nas linhas 53 à 58 e 60 à 62.

Especificação do Método de Seleção de Ação. Na implementação original o autor teve que se preocupar com codificar o método de seleção de ação, onde foram necessárias 14 linhas de código (figura 14 linha 18 e linhas 29 à 42). Com o uso da extensão, apenas uma linha precisou ser escrita (figura 15 linha 15).

Aprendizagem Mesmo em diversos casos sendo necessário mais linhas para implementar um mesmo componente do *Q-Learning* utilizando a extensão do quem sem a utilização da mesma, a complexidade de implementar o *Q-Learning* sem a extensão aparentemente é maior. Isso devido ao fato de que com o uso da extensão o desenvolvedor não precisa se preocupar com todo o fluxo do aprendizado. Das linhas 22 à 52 da figura 14 (implementação original) acontecem as seguintes operações: i) é iniciado um episódio; ii) uma ação é escolhida para o estado atual de acordo com o método de seleção de ação; iii) a ação é executada; vi) é observado a recompensa de estar no novo estado; e, por fim, v) a tabela Q é atualizada. Todas essas operações, com o uso da extensão, acontecem com apenas o uso de um comando, chamado na linha 24 da figura 15. Porém, para que isso seja pos-

sível, foi necessário ao programador nas linhas 11 à 17 da figura 15 informar todos os componentes do *Q-Learning*, e para isso diversas *procedures* e *reporters* tiveram que ser criados, o que levou ao aumento das linhas de código.

Quando um desenvolvedor precisa implementar o algoritmo *Q-Learning* ele precisa ser capaz de identificar os componentes do algoritmo para poder modelar corretamente o cenário, e, precisa também, conhecer bem o fluxo do algoritmo para poder implementá-lo sem erros. Com o uso da extensão o desenvolvedor pode se preocupar apenas com a primeira parte, sendo a segunda total responsabilidade da extensão através do método `learning`. Além disso o desenvolvedor não precisa se preocupar com a estrutura de dados utilizada para armazenar a tabela *Q*.

Figura 14 – Implementação do cenário *Maze* sem o uso da extensão.

```
patches-own[reward Qlist]
breed [ Walkers Walker ]
globals[episode]
Walkers-own [Hlist reward-list]

to setup
    clear-all
    ask patches [set Qlist [0 0 0 0]]
    ask patch -4 -4 [
        sprout-Walkers 1 [
            set Hlist [90 180 270 0]
        ]
    ]
end

to go
    let num-episodes 100
    let exploration-% 0
    let step-size 1
    let discount 1
    set episode 1
    while [episode <= num-episodes][
        ask walkers [set xcor -4 set ycor -4]
        while [[reward] of patch-here = 0][
            let Qnew 1
            let Qmax 0
            let dirp 0

            let rand random-float 1
            ifelse rand <= exploration-% [
                let dir one-of Hlist
                set dirp position dir Hlist
                set Qmax max Qlist
                set Qnew item dirp Qlist
            ][
                while [Qnew != Qmax][
                    let dir one-of Hlist
                    set dirp position dir Hlist
                    set Qmax max Qlist
                    set Qnew item dirp Qlist
                ]
            ]
            set heading item dirp Hlist
            let r [reward] of patch-ahead 1
            let QQnew max [Qlist] of patch-ahead 1
            set Qnew Qnew + step-size * (r + discount * QQnew - Qnew)
            set Qnew precision Qnew 3
            set Qlist replace-item dirp Qlist Qnew
            fd 1
        ]
    set episode episode + 1
  ]
end
```

Fonte: Roop (2006)

Figura 15 – Implementação do cenário *Maze* utilizando a extensão

```
extensions[qlearningextension]
patches-own[reward]
breed[Walkers Walker]

to setup
    clear-all
    ask patch -4 -4 [
        sprout-Walkers 1
    ]
    ask Walkers [
        qlearningextension:state-def ["xcor" "ycor"]
        (qlearningextension:actions [goUp] [goDown] [goLeft] [goRight])
        qlearningextension:end-episode [isEndState] resetEpisode
        qlearningextension:reward [rewardFunc]
        qlearningextension:action-selection "random-normal" [0]
        qlearningextension:learning-rate 1
        qlearningextension:discount-factor 1
    ]
end

to go
    ask Walkers [
        if qlearningextension:episode <= 100 [
            qlearningextension:learning
        ]
    ]
end

to-report rewardFunc
  report [reward] of patch-here
end

to goUp
    set heading 0
    fd 1
end

to goDown
    set heading 180
    fd 1
end

to goLeft
    set heading 270
    fd 1
end

to goRight
    set heading 90
    fd 1
end

to-report isEndState
  if [reward] of patch-here != 0 [
    report true
  ]
  report false
end

to resetEpisode
  setxy -4 -4
end
```

Fonte: O autor.

5 CONCLUSÃO

Nesse trabalho foi desenvolvida uma extensão para a plataforma NetLogo que fornece funções para auxiliar a incorporação do comportamento de aprendizagem por reforço *Q-learning* em simulações com agentes. As funções implementadas permitem aos desenvolvedores de simulações com agentes especificar: estados, ações, recompensa, método de seleção de ação, cláusula de fim de episódio, procedimento de reinício de episódio, taxa de aprendizagem e fator de desconto. A criação da tabela Q, iniciação de um episódio, escolha de uma ação, execução da ação, observação da recompensa no novo estado e atualização da tabela Q é de responsabilidade da extensão. A extensão fornece também alguns comandos para depuração, que permitem acompanhar a atualização da tabela Q, visualizar a tabela Q, obter o episódio atual e o valor atual de ε.

Com a extensão pronta, buscou-se comprovar a corretude da implementação do método *Q-Learning* e analisar as características de seu código. Para isso, duas análises foram feitas. Na primeira desenvolveu-se uma simulação de um cenário clássico de aprendizagem utilizando a extensão (*cliff walking*), e a implementação foi executada 10 vezes para obtenção de uma média e desvio padrão da tabela Q. Observou-se nessa análise que a extensão conseguiu obter a política ótima, evidenciando a corretude da implementação da extensão. A segunda análise buscou comparar as características do código de uma mesma simulação implementada utilizando a extensão e sem a utilização da mesma. Nessa análise foi possível observar que apesar de a implementação sem extensão possuir menos linhas de código, com o uso da extensão a implementação é menos complexa já que a extensão controla todo o fluxo do aprendizado.

Com relação aos trabalhos correlatos, o diferencial deste trabalho é que foi desenvolvida uma nova extensão para a plataforma NetLogo com o foco em oferecer aos desenvolvedores de simulações com agentes uma maneira de incorporar o algoritmo *Q-Learning* em suas simulações. A extensão foi disponibilizada para uso da comunidade através do *Extension Manager* que permite aos usuários adicionarem a extensão aos seus projetos com apenas 4 cliques.

A análise das características do código fonte foi feita apenas com uma simulação, e tomando como base um código fonte já existente. Para coletar evidências mais robustas dos benefícios da extensão, seria necessário realizar um experimento com desenvolvedores, onde eles fariam a implementação com e sem a extensão. Dessa maneira seria possível medir o esforço (tempo) de desenvolvimento com cada uma das abordagens. A realização deste experimento com desenvolvedores é uma sugestão de trabalho futuro.

Uma outra sugestão de trabalho futuro diz respeito ao fluxo de aprendizado. Atualmente todo o fluxo de aprendizado é controlado por apenas um comando da extensão, o `learning`.

Essa abordagem foi adotada pois permite ao desenvolvedor se preocupar com menos detalhes do algoritmo. Ao quebrar este esse fluxo em mais comandos, a extensão se tornaria mais flexível, proporcionando mais liberdade ao desenvolvedor. Dessa maneira ele poderia ou não seguir o fluxo da extensão e decidir fazer coisas contrárias ao que a extensão com o comando `learning` faria, como por exemplo, tomar uma ação contrária ao método de seleção de ação.

Por fim, seria importante avaliar a extensão em outras simulações onde se utiliza a aprendizagem por reforço. Desse modo seria possível garantir seu funcionamento nos mais variados cenários e condições atestando então a facilidade de uso da extensão.

REFERÊNCIAS

EL-TANTAWY, S.; ABDULHAI, B. An agent-based learning towards decentralized and coordinated traffic signal control. In: IEEE. **13th International IEEE Conference on Intelligent Transportation Systems**. Funchal – Madeira Island, Portugal, 2010. p. 665–670.

GALÁN, J. M.; IZQUIERDO, L. R.; IZQUIERDO, S. S.; SANTOS, J. I.; OLMO, R. D.; LÓPEZ-PAREDES, A.; EDMONDS, B. Errors and artefacts in agent-based modelling. **Journal of Artificial Societies and Social Simulation**, v. 12, n. 1, p. 1, 2009.

IZQUIERDO, L. R.; OLARU, D.; IZQUIERDO, S. S.; PURCHASE, S.; SOUTAR, G. N. Fuzzy logic for social simulation using netlogo. **Journal of Artificial Societies and Social Simulation**, v. 18, n. 4, p. 1, 2015.

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

KLÜGL, F. A validation methodology for agent-based simulations. In: **Proceedings of the 2008 ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2008. (SAC '08), p. 39–43. ISBN 978-1-59593-753-7. Disponível em: <http://doi.acm.org/10.1145/1363686.1363696>.

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

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

RAMOS, G. de. O.; LEMOS, L. L.; BAZZAN, A. L. C. Developing a python reinforcement learning library for traffic simulation. In: BRYS, T.; HARUTYUNYAN, A.; MANNION, P.; SUBRAMANIAN, K. (Ed.). **Proceedings of the Adaptive Learning Agents Workshop 2017 (ALA2017)**. São Paulo: [s.n.], 2017. (ALA2017). Disponível em: <http://www.inf.ufrgs.br/maslab/pergamus/pubs/Ramos+2017ala.pdf>.

ROOP, J. **Reinforcement Learning Maze**. NetLogo User Community Models, 2006. Aerospace Systems Design Laboratory (ASDL), Georgia Institute of Technology. Disponível em: <http://ccl.northwestern.edu/netlogo/models/community/Reinforcement\\%20Learning\\%20Maze>.

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

SANTOS, F. **Model driven agent based simulation development**. Tese de Doutorado — Universidade Federal do Rio Grande do Sul (UFRGS), Janeiro 2019.

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

TISUE, S.; WILENSKY, U. Netlogo: Design and implementation of a multi-agent modeling environment. In: **Proceedings of the Agent 2004 Conference on Social Dynamics: Interaction, Reflexivity and Emergence**. Chicago – Illinois, USA: [s.n.], 2004. p. 7–9.

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

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: theory and practice. **The Knowledge Engineering Review**, Cambridge University Press, v. 10, n. 2, p. 115–152, 1995.