# An Automated Approach for Verifying Learned Policies of Q-Learning Agents in NetLogo

**Mateus Rissardi**[1], **Fernando Santos**[1]

[1]Departamento de Engenharia de Software
Universidade do Estado de Santa Catarina (UDESC) – Ibirama, SC - Brasil

mrissardi01@gmail.com, fernando.santos@udesc.br

***Abstract.*** *The use of Reinforcement Learning (RL) algorithms in Agent Based Models (ABM) is of great practical interest to simulate complex environments. An extension of the NetLogo tool is available to simplify the use of RL in ABMs. Despite their benefits, the verification of the agent's learning in this extension remains a challenge, as it requires manual inspections in the data structure used by the RL algorithms. This paper proposes an automated verifier of the policy learned by the agent. The verifier compares the policy learned by the agent with a desired policy, informed by the developer, and indicates if there is divergence. The use of the verifier is demonstrated through Cliff Walking and Labyrinth ABMs.*

***Resumo.*** *O uso de algoritmos de Aprendizado por Reforço (RL) em Modelos Baseados em Agentes (MBA) é de grande interesse prático para simular ambientes complexos. Uma extensão da ferramenta NetLogo está disponível para simplificar o uso de RL em MBA. Apesar de seus benefícios, a verificação do aprendizado dos agentes nessa extensão permanece um desafio, pois exige inspeções manuais nas estruturas de dados utilizadas pelos algoritmos de RL. Este artigo propõe um verificador automatizado da política aprendida pelo agente. O verificador compara a política aprendida pelo agente com uma política desejada, informada pelo desenvolvedor, e indica se há divergência. O uso do verificador é demonstrado por meio dos MBAs Cliff Walking e Labirinto.*

## 1. Introdução

Modelos Baseados em Agentes (MBA) são um artefato de grande utilidade para a exploração de sistemas complexos, como fenômenos naturais e sociais. As MBAs em sua essência são uma implementação virtual de um ambiente fictício, onde nele são habitados agentes atuadores. Dentre diversas ferramentas reconhecidas para o desenvolvimento de tais modelos, a ferramenta NetLogo [Wilensky 1999] é popular por sua facilidade e intuitividade, sendo amplamente utilizada para fins educacionais e pesquisas científicas.

O uso de Aprendizado por Reforço (RL) tem se mostrado fundamental na construção de agentes inteligentes em sistemas multiagentes, permitindo que estes aprendam comportamentos autônomos por meio de interações de tentativa e erro em ambientes complexos. As aplicações de RL no cotidiano abrangem diversas áreas de conhecimento. Alguns de seus usos são encontrados nos algoritmos de recomendação em sites como Youtube [Chen et al. 2019], no treinamento de robôs automatizados [Polydoros and Nalpantidis 2017] e no auxílio para profissionais da saúde

na escolha de medicações para pacientes com estados de saúde de rápida mudança [Sezgin and Özkan 2013].

A implementação de algoritmos de RL em MBAs é complexa, exigindo ao modelador grande conhecimento técnico sobre o funcionamento de tais algoritmos. Foi desenvolvido para a ferramenta NetLogo uma extensão de RL, criada inicialmente por [Kons 2019] e posteriormente refatorada por [Bazzanella et al. 2024]. Esta extensão propõe facilitar a implementação e uso de algoritmos de RL, disponibilizando comandos que simplificam a configuração dos algoritmos e auxiliando na criação de agentes inteligentes.

Contudo, garantir que esses agentes tenham convergido para uma política de ação adequada é um desafio recorrente. Embora a extensão de RL facilite a implementação desses modelos, ela carece de mecanismos integrados para a verificação automática do aprendizado, limitando-se, atualmente, à exposição de valores numéricos encontrados na política de ação, o que torna a verificação humana complexa e propensa a erros.

Este artigo propõe um mecanismo de verificação de aprendizagem do agente, integrado à extensão de RL existente. O objetivo é permitir que o desenvolvedor verifique, de forma simples e controlada, se a política aprendida pelo agente corresponde à política desejada, superando a dificuldade de interpretar dados brutos. Para assegurar que os agentes aprenderam as políticas corretamente, o comando realiza uma comparação direta entre a política de ação gerada e uma política de referência fornecida externamente pelo desenvolvedor. Os resultados obtidos utilizando o comando apontam a sua eficácia como meio para verificar o conhecimento aprendido pelos agentes.

## 2. Fundamentação Teórica

Nesta seção, serão explicados os principais tópicos encontrados neste trabalho, sendo esses os conceitos de Modelagem Baseada em Agentes e Aprendizado por Reforço, a ferramenta NetLogo, a extensão de Aprendizado por Reforço utilizada e, por fim, os princípios da Verificação de Software.

### 2.1. Modelagem Baseada em Agentes

Para um melhor entendimento sobre o que é modelagem baseada em agentes é necessário definir primeiro os conceitos de modelo e agentes. A definição de um modelo, em sua essência, se refere a uma representação simplificada, padrão ou referência usado como base para a criação de sistemas, conceitos ou objetos. Agentes são definidos por tudo que é capaz de observar o seu ambiente com o uso de sensores e de agir nesse ambiente por meio de seus atuadores [Russell and Norvig 2022].

A modelagem baseada em agentes é uma abordagem para a modelagem de sistemas compostos por agentes interativos autônomos, oferecendo formas fáceis de modelar os comportamentos individuais de agentes e como seus comportamentos afetam o ambiente e outros agentes [Macal and North 2014]. Em Modelos Baseado em Agentes (MBA) é possível representar computacionalmente os elementos da realidade por meio de tais agentes, impondo regras simples, mas eficientes na transferência do conhecimento do mundo real para o modelo [Silva et al. 2025].

Por meio dos MBAs, é possível representar e simular fenômenos naturais, sociais e computacionais que em grande parte seriam complexos de replicar em um ambiente

real. Com o seu uso, os MBAs se tornam uma ferramenta essencial para diversas áreas de pesquisa, auxiliando pesquisadores a simular ambientes e observar os resultados com facilidade [Silva et al. 2025].

### 2.2. NetLogo

A ferramenta NetLogo oferece uma linguagem de programação multi-agente e um ambiente de modelagem programável para simular fenômenos sociais e naturais [Wilensky 1999]. Os modeladores conseguem dar instruções para centenas ou milhares de agentes operando de forma independente. Isso torna possível a exploração de conexões entre indivíduos em nível microscópico e analisar o padrão de suas interações em um nível macroscópico.

A NetLogo foi desenvolvida com o objetivo em mente de exigir baixo conhecimento prévio sobre modelagem baseada em agentes, mas ainda possuindo um grande leque de possibilidades sobre o que pode ser feito com a ferramenta *(Low threshold, High ceiling)*, permitindo que pessoas de diferentes níveis de conhecimento consigam utilizar a ferramenta, mas ainda que sejam desenvolvidos modelos de níveis avançados.

A NetLogo permite também que desenvolvedores criem suas próprias extensões para a ferramenta, assim possibilitando que o mesmo escreva novos comandos que antes não eram possíveis ou encontrados na sua linguagem de programação. A API da NetLogo, disponível em linguagens como Scala e Java, permite a troca de informações direta entre o modelo criado a partir da NetLogo e os códigos externos escritos pelo desenvolvedor.

### 2.3. Aprendizado por Reforço e o Algoritmo Q-Learning

O Aprendizado por Reforço *(Reinforcement Learning — RL)* se refere a um dos ramos de Aprendizado de Máquina *(Machine Learning)* no qual um agente, que está diante de um problema, deve aprender um comportamento por meio de interações de tentativa e erro dentro de um ambiente dinâmico [Kaelbling et al. 1996].

Diferente do Aprendizado Supervisionado e Não-Supervisionado, o RL não precisa de um conjunto de dados pré-existente. O agente deve interagir com o ambiente e por meio das recompensas, resultante de suas ações, deve descobrir um padrão de ações desejado [Ris-Ala 2023], padrão esse conhecido como Política *(Policy)*. Uma política ideal, por sua vez, é a estratégia de ações que auxilia o agente a adquirir o melhor resultado.

Os algoritmos de RL buscam maximizar recompensas ao longo do tempo usando duas políticas: a de comportamento (para explorar o ambiente) e a alvo (para definir a ação ideal). De forma geral, o objetivo central dos algoritmos de RL é encontrar uma política que maximize as recompensas acumuladas ao longo do tempo [Nunes et al. 2023].

Algoritmos como o *Q-Learning* tem como sua forma de aprendizado avaliar qual é a melhor ação a ser tomada levando em consideração seu estado atual [Watkins and Dayan 1992]. Ele realiza isso por meio de uma função de estado-ação conhecida como *Q(S,A)*, que avalia tanto a sua recompensa imediata, ao realizar uma ação A em um estado *S*, quanto as recompensas a longo prazo, definindo o quão benéfico é chegar no estado *S'* devido a uma ação *A*. O aprendizado do agente é realizado de forma iterativa, sendo necessário diversos episódios de tentativa e erro para aprimorar seu conhecimento.

A fórmula de atualização dos valores *Q(S,A)* apresenta a seguinte configuração:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \quad (1)$$

onde,

- S é o estado atual.
- A é a ação tomada pelo agente.
- S' é o próximo estado a qual o agente se move.
- A' é a melhor ação encontrada no estado S'.
- R é a recompensa por tomar a ação A no estado S.
- $\gamma$ *(gamma)* é o fator de desconto que equilibra as recompensas imediatas com as recompensas futuras.
- $\alpha$ *(alpha)* é a taxa de aprendizado que determina o quanto uma nova informação afeta o antigo valor *Q(S,A)*.

Os valores *Q(S,A)* são armazenados dentro de uma Tabela-Q *(Q-Table)*. Uma vez que essa tabela seja preenchida e aprendida ela acaba por se tornar a política de ação do agente [Ris-Ala 2023].

### 2.4. Extensão de Aprendizado por Reforço do NetLogo

Criado inicialmente por [Kons 2019], a extensão tem como objetivo facilitar a implementação do algoritmo *Q-Learning* para a NetLogo, auxiliando na construção de agentes inteligentes. A extensão foi inicialmente desenvolvida utilizando a linguagem de programação Scala, uma das linguagens disponíveis para o desenvolvimento de extensões na NetLogo, e em sua versão inicial, possui somente o algoritmo *Q-Learning* para o aprendizado dos agentes. O estudo de [Bazzanella and Santos 2021] evidenciou que a extensão reduz o esforço necessário para utilizar RL em MBAs.

Refatorada anos depois por [Bazzanella et al. 2024], a extensão recebeu uma série de implementações e alterações, cujo expandiram o escopo inicial da extensão. Uma das mudanças de maior impacto é a introdução do uso da biblioteca de RL conhecida como BURLAP e a troca da linguagem de programação usada no desenvolvimento da extensão, a qual foi substituído pela linguagem Java. A BURLAP é uma biblioteca Java usada no desenvolvimento de modelos de agentes inteligentes, a qual disponibiliza uma série de algoritmos de aprendizado por reforço e de planejamento para o seu uso nos agentes [MacGlashan et al. 2018].

A extensão disponibiliza uma série de comandos, que propõe viabilizar a implementação dos algoritmos de RL de forma prática, representados na figura 1. A estrutura escolhida para os comandos de configuração auxilia nesse aspecto, exigindo somente os parâmetros necessários para configuração do algoritmo. Esses comandos tem como função principal resgatar os dados importantes do modelo, que por sua vez são passados para os parâmetros dos algoritmos encontrados na biblioteca BURLAP.

Atualmente, devido a forma como a extensão foi estruturada, a única maneira disponível de realizar a verificação da política de ação é analisando manualmente os dados encontrados pelo comando de retorno de dados, o que acaba por ser uma alternativa lenta e propensa a erros.

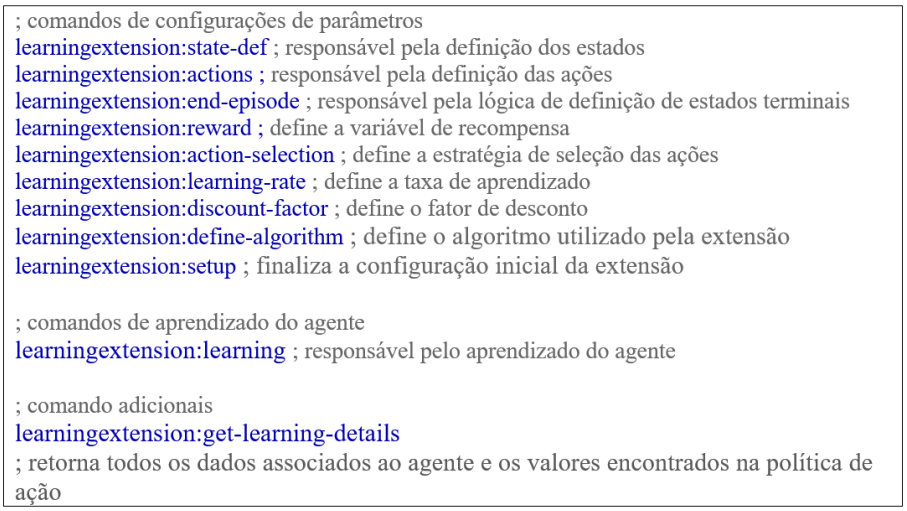



**Figura 1. Comandos disponíveis pela extensão. Fonte [Bazzanella et al. 2024]**

### 2.5. Verificação de Software

A realização de testes se configura como uma parte fundamental no desenvolvimento de software, e com o seu uso frequente durante o desenvolvimento ele ajuda na prevenção e correção de bugs, auxiliando na longevidade do código ao longo prazo. Entretanto os testes se referem somente a uma parte de um processo maior conhecido por verificação e validação de software.

Os processos de verificação e validação visam analisar se o software em desenvolvimento cumpre com o que foi estipulado, garantindo que os requisitos propostos sejam entregues e as expectativas do cliente sejam alcançadas [Sommerville 2018]. Dentro desse contexto, a verificação de software possui uma abordagem de análise voltada aos cumprimentos dos requisitos funcionais e não funcionais, avaliando se o software está funcionando conforme o esperado. O processo de verificação de software utiliza de inspeções e testes unitários para realizar a sua análise.

A complexidade de garantir uma verificação adequada, no entanto, é influenciado com o tamanho do projeto. Em códigos de baixa escala, realizar os testes manualmente para verificação ainda consegue ser prático, entretanto quando se trata de sistemas maiores que necessitam de uma série de testes isso se torna um problema considerável a ser resolvido. Para viabilizar o processo de verificação, um método utilizado é a implementação de testes automatizados. Os testes automatizados se referem no uso de ferramentas, frameworks ou scripts, para realizar uma série de testes sem a necessidade de intervenção humana para fazer a sua verificação [Bernardo and Kon 2011]. O uso de tal automatização auxilia na realização de testes, que verificados de forma manual levariam horas, em minutos ou até instantes.

## 3. Trabalhos Relacionados

Diferente da área de testes com agentes de RL, testes efetuados em agentes ainda é uma prática incomum e pouco abordada, onde fatores como a inconsistência de resultados, imprevisibilidade dos estados e incoerências da política gerada acaba por ser um fator de grande dificuldade para pesquisadores realizarem testes confiáveis [Sunba et al. 2026].

Na pesquisa de [Tappler et al. 2022] foram constatados problemas similares, e a partir disso foi proposto um framework de testes baseado em pesquisa que avalia tanto o desempenho quanto a segurança dos agentes de aprendizado por reforço profundo *(deep RL)*. O foco principal de seus testes se volta às questões de segurança nas ações do agente, verificando se o agente está tomando as decisões corretas em momentos críticos, e de performance, colocando o agente em uma variedade de estados pouco visitados para analisar o seu comportamento.

Em outro estudo realizado por [Mazouni et al. 2024] é abordado que os testes deveriam ir além de encontrar o número máximos de falhas, mencionando a importância de encontrar falhas diversas e informativas para a compreensão do comportamento do agente. A solução proposta pela pesquisa é de formular testes de política utilizando algoritmos de *Quality Diversity (QD)*, que buscam encontrar um conjunto de soluções que sejam simultaneamente de alta qualidade e completamente diversas [Pugh et al. 2016] com o intuito de encontrar falhas em situações que o agente normalmente não encontraria.

Ambas as pesquisas citadas procuram analisar o comportamento do agente e explorar por possíveis estados onde o agente possa falhar na escolha de sua ação. Esta pesquisa diferencia-se abordando um teor mais comparativo e analítico para seus testes, diferente das pesquisas apresentadas anteriormente, onde seu foco está diretamente associado à política de ação gerada pelo agente, a qual ela vai ser analisada diretamente com a política gerada pela extensão.

Esta pesquisa utilizou como base de seu estudo a extensão de aprendizado por reforço do NetLogo. A extensão propõe facilitar o uso de algoritmos de aprendizado por reforço em MBAs, disponibilizando comandos que permitem a fácil configuração de seu algoritmo, e por fim, simplificando o processo de criação de agentes inteligentes.

Um problema relevante encontrado em ambas as versões da extensão é a ausência de meios para realizar uma verificação sobre o aprendizado do agente, possuindo apenas um comando para retornar os valores encontrados na *Q-Table*. A falta de um comando verificador que realize a análise da política aprendida pelo agente motivou a realização desta pesquisa.

## 4. Desenvolvimento do Verificador de Aprendizagem

O comando de verificação propõe em realizar uma análise da política de ação gerada pela extensão e, com base em uma política desejada, analisar se as ações aprendidas correspondem as ações desejadas pelo desenvolvedor. Foi escolhida essa abordagem devido a dificuldade e complexidade de verificar a política gerada utilizando somente os valores numéricos encontrados na Q-Table.

Esse método de testes proporciona uma análise simplificada, onde o desenvolvedor pode avaliar diretamente a política de ação realizando uma comparação direta com as ações geradas pela Q-Table e as ações desejadas. Para o comando realizar essa ação é necessário como parâmetro o caminho de um arquivo CSV, um formato de texto simples usado para armazenar dados em tabela, que representa a política desejada pelo desenvolvedor. A chamada do comando na NetLogo é apresentada na figura 2.

A política desejada deve conter todos os estados alcançáveis no modelo em conjunto de sua ação desejada, podendo ser ou não idêntica a política ideal. A tabela segue

```
; comando de verificação
print learningextension:verify-learning"C:\\Users\\politica_desejada.csv"
; separação das pastas deve ser realizado com duas contrabarras "\\" no Windows
```

**Figura 2. Comando de verificação implementado na NetLogo**

um formato contendo pelo menos duas colunas, apresentado na figura 3, onde a primeira coluna representa os estados acessíveis pelo agente, sem contabilizar estados terminais, e a segunda em diante representa os nomes das ações desejadas. Vale notar que os nomes das ações devem corresponder aos nomes encontrados no modelo NetLogo. O comando permite que seja associado na tabela até duas ações para um estado, essa abordagem foi adotada devido a casos específicos em que não existe uma única ação desejada em um determinado estado. Em casos raros que qualquer ação pode ser desejada em um estado é possível adicionar a ação "*" na tabela, que deve aceitar qualquer resposta gerada pela extensão.

```
State;Action
0-0;goUp
0-1;goRight
0-2;goDown;goRight
2-1;*
```

**Figura 3. Exemplo de tabela da política desejada**

Após receber o caminho do arquivo, o verificador dispara o processo da análise dos dados, onde ele realiza as seguintes etapas:

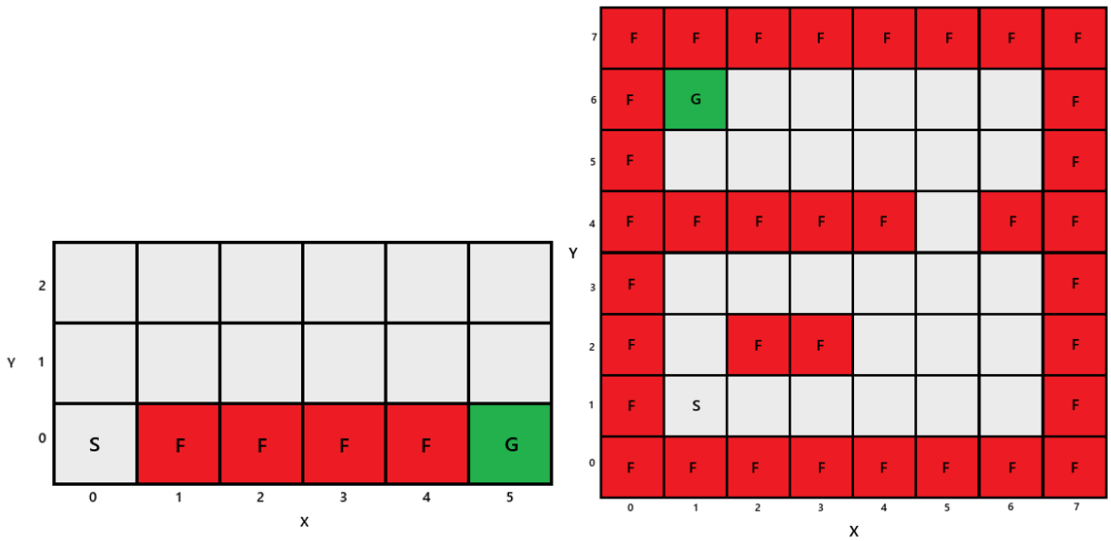
- **Validação dos Dados:** Ao realizar a chamado do método, o verificador faz a busca pelo arquivo no caminho mencionado, e ao encontrar, efetua uma validação de compatibilidade dos dados informados pela tabela da política desejada. Caso ocorrer algum erro em alguma das etapas o comando retorna uma mensagem de erro para o terminal da NetLogo.
- **Filtragem e Mapeamento** Nessa etapa é efetuado a filtragem dos dados obtidos da tabela da política desejada e da política gerada pela extensão, armazenando e mapeando-os em uma variável Map que permita realizar as etapas seguintes.
- **Busca Iterativa:** Com os dados mapeados corretamente, o comando acessa um estado mapeado na política gerada pela extensão e realiza uma busca pelo seu estado correspondente da política desejada. Esse processo é realizado, após uma iteração de verificação, para cada estado encontrado na política gerada. Caso nenhuma correspondência seja encontrada, o verificador deve disparar um erro.
- **Verificações das Ações:** Após encontrar o seu estado correspondente em ambas as políticas, o comando realiza a comparação entre as ações encontradas nas políticas, retornando uma mensagem de alerta caso as ações não sejam equivalentes. Essa operação é ignorada caso a ação desejada se refere a uma ação qualquer "*". A mensagem de alerta deve conter o estado em que houve a divergência, a ação esperada e a ação encontrada na *Q-Table*.

Ao final da execução, o comando deve retornar o resultado da verificação das políticas de ações, apresentando possíveis divergências encontradas durante o processo de verificação.

## 5. Modelagem do Ambiente de Testes

Para testar o funcionamento do novo comando era necessário modelar um ambiente confiável e determinístico, isso veio devido a necessidade de verificar, de forma concreta, o retorno da política gerada. Foi utilizado então como base de testes dois modelos, o primeiro sendo o clássico problema do *Cliff Walking* [Sutton et al. 1998] e o segundo sendo um problema de labirinto. Ambos os modelos apresentam problemas idênticos, um agente precisa chegar do estado (S) para um estado (G) sem passar pelos estados (F), o que os diferencia é em como os ambientes são estruturados.

O modelo do *Cliff Walking*, apresentado na figura 4(a), visa uma abordagem de estrutura de ambiente mais linear e condensada, possuindo somente um único caminho considerado ideal. Em contrapartida, o ambiente do labirinto, apresentado na figura 4(b), possui uma estrutura mais variada, criando diversas possibilidades de rotas a serem seguidas que por sua vez influenciam no número de políticas ideais encontradas no modelo. Foi escolhida essa estrutura dos modelos para garantir que o comando tenha a capacidade de verificar ambientes com políticas de ações diversificadas, onde seja possível que mais de uma ação seja desejada.




**(a)** *Cliff Walking* **(b)** Ambiente Labirinto

**Figura 4. Modelo dos ambientes utilizados para testes**

As ações disponibilizadas para os agentes em ambos os modelos são de mover para: esquerda, cima, direita e baixo. Ao se movimentar, o agente consegue encontrar três tipos de estado: estados comuns, estados de falha (F) e o estado desejado (G). Cada estado possui uma variável de recompensa, usada para reforçar o aprendizado do agente sobre o benefício de estar em tal estado. Os estados comuns possuem um valor de recompensa negativo de -0,04. Os estados de falha e o estado desejado são terminais, que ao serem

alcançados retornam o agente para o estado inicial (S), possuindo respectivamente uma recompensa de -1 e 1.

Os parâmetros das variáveis de aprendizado foram selecionados com base no artigo feito por [Bazzanella et al. 2024], possuindo os seguintes valores para os modelos: $\alpha = 0,1$, $\gamma = 0,3$, 1500 episódios de aprendizagem. A decisão dos parâmetros de aprendizado foi realizada de tal forma para que os resultados da política sejam similares aos obtidos por [Bazzanella et al. 2024], além disso eles promovem um caráter de comportamento exploratório, sendo essencial para o desenvolvimento de uma política de ação que se aproxima a política ideal.

## 6. Resultado do Verficador

Para realizar a avaliação da efetividade do comando, foram efetuados testes utilizando os modelos do *Cliff Walking* e de labirinto. Para cada instância de teste realizado em seus respectivos modelos, foram utilizados uma política de ação preenchida manualmente no formato de arquivo CSV, onde a figura 5(a) representa o modelo *Cliff Walking* e a 5(b) representando o modelo labirinto.

| **State;Action** |
|---|
| 0-0;goUp |
| 0-1;goRight |
| 0-2;goDown;goRight |
| 2-1;goRight |
| 1-1;goRight |
| 1-2;goRight;goDown |
| 2-2;goDown;goRight |
| 3-1;goRight |
| 3-2;goDown;goRight |
| 4-1;goRight |
| 4-2;goDown;goRight |
| 5-1;goDown |
| 5-2;goDown |

**(a)** *Cliff Walking*

| **State;Action** |
|---|
| 1-1;goUp;goRight |
| 1-2;goUp |
| 1-3;goRight |
| 1-5;goUp |
| 2-1;goRight |
| 2-3;goRight |
| 2-5;goUp;goLeft |
| 2-6;goLeft |
| 3-1;goRight |
| 3-3;goRight |
| 3-5;goUp;goLeft |
| 3-6;goLeft |
| 4-1;goUp;goRight |
| 4-2;goUp;goRight |
| 4-3;goRight |
| 4-5;goUp;goLeft |
| 4-6;goLeft |
| 5-1;goUp |
| 5-2;goUp |
| 5-3;goUp |
| 5-4;goUp |
| 5-5;goUp;goLeft |
| 5-6;goLeft |
| 6-1;goUp;goLeft |
| 6-2;goUp;goLeft |
| 6-3;goLeft |
| 6-5;goUp;goLeft |
| 6-6;goLeft |

**(b)** Ambiente Labirinto

**Figura 5. Políticas de ações desejadas em formato CSV**

Conforme o retorno apresentado pela figura 6, o verificador encontrou como resultado nenhuma divergência nas políticas de ações analisadas. Isso implica na ideia de

que o agente aprendeu corretamente a política de ação desejada pelo desenvolvedor. No resultado apresentado do ambiente *Cliff Walking* evidencia-se o funcionamento do verificador, que identificou que a política aprendida pelo agente corresponde à política desejada pelo desenvolvedor.

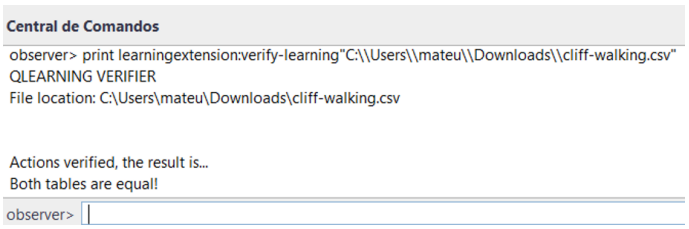



**Figura 6. Retorno do comando no ambiente *Cliff Walking***

Durante o uso do comando no caso de testes do labirinto, é possível notar o retorno de uma divergência encontrada entre a políticas, apresentadas na figura 7, demonstrando que o conhecimento apreendido pelo agente divergiu do conhecimento desejado pelo desenvolvedor. Com essa informação do retorno, é possível notar o funcionamento do comando verificador, que verificou e apontou a divergência encontrada entre a política aprendida correspondente à política desejada.

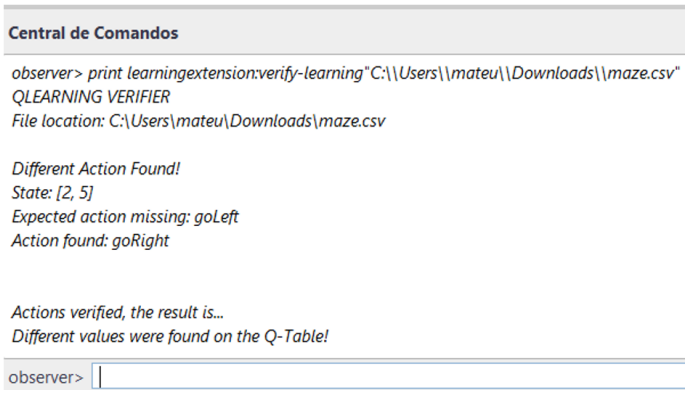



**Figura 7. Retorno do comando no ambiente Labirinto**

Os resultados obtidos pela execução do comando demonstram a efetividade do uso da ferramenta, viabilizando uma compreensão sobre o aprendizado efetuado pelo agente que antes não era possível na extensão. Os modelos estão disponíveis em repositório online[1].

## 7. Conclusão

Neste artigo foi proposto a implementação de uma ferramenta que possibilitasse a análise simplificada sobre o aprendizado do agente, efetuado por meio da extensão de aprendizado por reforço da NetLogo. Seu resultado nos casos de teste demonstra que o comando consegue realizar essa análise de uma forma efetiva, simplificada, controle do desenvolvedor. A limitação no desenvolvimento do comando necessitou que o desenvolvimento do mesmo tomasse uma abordagem simplificada, mas que ainda resolvesse o problema inicial proposto.

[1] https://github.com/agentbasedsimulations/2026-wesaac-netlogo-verify-qlearning

O comando obteve resultados satisfatórios em sua execução, entretanto a sua implementação ainda apresenta uma margem para melhorias e alterações. Algumas das limitações presentes são: a ausência de verificação para agentes que utilizam dos algoritmos *Sarsa(λ)* e *Actor-Critic*, presentes na extensão do NetLogo, a limitação da quantidade de ações permitidas pela tabela da política desejada, a qual apenas permite até duas ações, e por fim o uso da tabela CSV traz uma dificuldade para o uso do comando, exigindo um trabalho manual elevado para preenchê-la em casos de modelos maiores.

Devido a alguns pontos de melhoria necessários, o principal sendo referente ao preenchimento da tabela de política desejada, o comando ainda deve sofrer uma refatoração futura. Para futuros trabalhos com o comando será descartado o uso de uma política de ação externa, sendo substituído por uma política de ação gerada com um algoritmo de planejamento. Para cumprir com os requisitos dessa melhoria poderão ser utilizados os algoritmos disponíveis na biblioteca BURLAP[2]. Essa abordagem permite que o desenvolvedor realize testes do aprendizado do agente realizando somente a chamada do comando, sendo descartado a necessidade de um preenchimento manual da política, que por muitas vezes geram frustrações por sua natureza extensa.

## Referências


Bazzanella, E., Barros, M., and Santos, F. (2024). Refatoração da extensão netlogo de aprendizagem por reforço para integração com a biblioteca burlap. In *Anais do XVIII Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 51–62, Porto Alegre, RS, Brasil. SBC.

Bazzanella, E. and Santos, F. (2021). Does a q-learning netlogo extension simplify the development of agent-based simulations? In *Anais do XV Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 1–12, Porto Alegre, RS, Brasil. SBC.

Bernardo, P. C. and Kon, F. (2011). *Padrões de testes automatizados*. Tese de doutorado, Universidade de São Paulo (USP).

Chen, M., Beutel, A., Covington, P., Jain, S., Belletti, F., and Chi, E. H. (2019). Top-k off-policy correction for a reinforce recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 456–464, New York, NY, USA. Association for Computing Machinery.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Kons, K. (2019). *Biblioteca Q-Learning para desenvolvimento de simulações com agentes na plataforma NetLogo*. Trabalho de conclusão de curso, Universidade do Estado de Santa Catarina (UDESC).

Macal, C. and North, M. (2014). Introductory tutorial: Agent-based modeling and simulation. In *Proceedings of the winter simulation conference 2014*, pages 6–20. IEEE.

MacGlashan, J., Loftin, R., Littman, M. L., and Roberts, D. L. (2018). BURLAP: Brown-UMBC Reinforcement Learning and Planning. `burlap.cs.brown.edu`.


[2] `http://burlap.cs.brown.edu/`

Mazouni, Q., Spieker, H., Gotlieb, A., and Acher, M. (2024). Testing for fault diversity in reinforcement learning. In *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024)*, pages 136–146.

Nunes, R. H., de Jesus Toledo, B., Bello, C. C. C., de Andrade, G. O., Mariano, G. T., and Marques, R. G. (2023). Inteligência artificial e aprendizagem por reforço. *Revista CBTecLE*, 7(2):210–225.

Polydoros, A. S. and Nalpantidis, L. (2017). Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40.

Ris-Ala, R. (2023). *Fundamentos de Aprendizagem por Reforço*. Ed. do Autor, Rio de Janeiro.

Russell, S. J. and Norvig, P. (2022). *Inteligência Artificial: Uma Abordagem Moderna*. GEN LTC, Rio de Janeiro, 4 edition.

Sezgin, E. and Özkan, S. (2013). A systematic literature review on health recommender systems. In *2013 E-Health and Bioengineering Conference (EHB)*, pages 1–4. IEEE.

Silva, C. R., Mesquita, O., Araújo, E., and Mata, A. S. (2025). Uso da modelagem baseada em agentes no estudo de sistemas complexos. *Revista Brasileira de Ensino de Física*, 47:e20240464.

Sommerville, I. (2018). *Engenharia de Software*. Pearson, 10 edition.

Sunba, A., Hassine, J., and Ahmed, M. (2026). Testing reinforcement learning systems: A comprehensive review. *Journal of Systems and Software*, 231:112563.

Sutton, R. S., Barto, A. G., and Barto, A. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tappler, M., Córdoba, F. C., Aichernig, B. K., and Könighofer, B. (2022). Search-based testing of reinforcement learning. *arXiv preprint arXiv:2205.04887*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3):279–292.

Wilensky, U. (1999). NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. `http://ccl.northwestern.edu/netlogo/`.