

Universidade Federal do ABC

**Ferramenta Interativa de Visualização de Agentes em
Ambiente de Aprendizado por Reforço**

Projeto de Graduação em Computação III – PGC-III

Aluno

Igor Neres Trindade

RA: 11030416

igor.neres@aluno.ufabc.edu.br

Orientador

Prof. Dr. Harlen Costa Batagelo

harlen.batagelo@ufabc.edu.br

30 de junho de 2022

Resumo

Aprendizado de Máquina é uma área de pesquisa que busca possibilitar o aprendizado de sistemas computacionais sem uma programação explícita do conhecimento, através da análise de dados, identificação de padrões, e com o mínimo de intervenção humana. Com o rápido avanço da ciência e a crescente adesão do Aprendizado de Máquina na resolução de problemas modernos, se tornou comum a aplicação de técnicas cada vez mais sofisticadas, em especial as de Aprendizado por Reforço.

Neste projeto de Graduação em Computação, criamos uma ferramenta de visualização de agentes em ambiente de Aprendizado por Reforço, de maneira a introduzir algoritmos e conceitos fundamentais a um público que possua um conhecimento mínimo em computação.

Para a realização deste trabalho, foram consideradas funcionalidades de ferramentas já existentes. Além de considerar deficiências e limitações delas, o projeto contribuiu com funcionalidades inéditas, com foco na interatividade e nas possibilidades de exploração pelo usuário. Disponibilizamos também, através deste trabalho, todo o código implementado em linguagem moderna, para que possa ser utilizado para fins didáticos em conjunto com a documentação feita neste relatório.

Palavras-chave: Aprendizado de Máquina, Aprendizado por Reforço, Visualização, Simulação.

Abstract

Machine Learning is a research area that seeks to give computer systems the ability to learn new knowledge without being explicitly programmed with that knowledge, through data analysis, pattern recognition, and minimum human intervention. As a result of the recent scientific contributions and the increasing employment of this knowledge for solving modern problems, sophisticated techniques have become common, especially those related to Reinforcement Learning.

In this Computer Science Undergraduate final project, we developed a visualization tool for Reinforcement Learning to be used to introduce foundational algorithms and concepts to people with a minimum Computer Science knowledge.

For the project execution, we considered the experience provided by existing tools. Besides working on their limitations, we contributed with new functionalities while always considering the interactivity and content exploration by the user. This is intended to be an open source project so that the code implementation in a modern programming language can be used for learning purposes along with the documentation in this report.

Keywords: Machine Learning, Reinforcement Learning, Visualization, Simulation.

Sumário

1	Introdução	6
2	Conceitos básicos	9
2.1	Agentes, Ambientes e Racionalidade	9
2.2	Processo de Decisão de Markov	10
2.2.1	Estocasticidade	12
2.2.2	Recompensas	13
2.2.3	Políticas e Funções-valores	14
2.2.4	Otimalidade	15
2.3	<i>Gridworld</i>	17
2.4	Algoritmos de Programação Dinâmica	20
2.4.1	Algoritmo de Iteração de Política	20
2.4.2	Algoritmo de Iteração de Valor	22
2.5	Algoritmos de Diferença Temporal	23
2.5.1	Aprendizagem por experiência	24
2.5.2	SARSA	25
2.5.3	Aprendizagem-Q	26
3	Trabalhos correlatos	28
4	Objetivos	32
4.1	Objetivo geral	32
4.2	Objetivos específicos	32
5	Método	34
5.1	Plataforma de desenvolvimento <i>Unity</i>	34
5.2	Gerenciadores de estado	35
5.3	Modelagem do <i>MDP</i>	36
5.4	Distribuição do conteúdo	38
6	Resultados	39
6.1	Funcionalidades principais da ferramenta	39
6.1.1	Parâmetros	40
6.1.2	Modos de interação	41
6.1.3	Modos de visualização	42

6.2	Possibilidades de exploração	42
7	Conclusão	48
7.1	Contribuições	48
7.2	Limitações e trabalhos futuros	49

1 Introdução

A sociedade moderna convive com um alto grau de sofisticação tecnológica. Muito dessa evolução ocorreu nas últimas décadas, especialmente devido ao surgimento de estudos em Inteligência Artificial (Yarlagadda, 2018). Nesse tempo, a humanidade foi capaz de criar ferramentas que auxiliassem em suas atividades diárias, e de automatizá-las, produzindo as máquinas. Porém, a evolução dessas máquinas, mesmo as mais sofisticadas, teve como notória barreira o fator de inteligência (Mohammed *et al.*, 2016).

Diferentemente das pessoas, máquinas, por natureza, não tomam decisões conscientemente com base nos eventos ocorridos em seus arredores e limitam-se a um conjunto bem definido de instruções pré-programadas. Questionamentos sobre essa limitação na capacidade de pensar se consolidaram na segunda metade do século passado e possibilitaram o surgimento de estudos em Inteligência Artificial. Hoje, dentro desta grande área, está o Aprendizado de Máquina que tenta possibilitar o aprendizado das máquinas sem uma programação explícita deste comportamento (Mohammed *et al.*, 2016), através da análise de dados e da identificação de padrões capazes de automatizar a criação e aprimoramento de modelos analíticos com o mínimo de intervenção humana.

Os principais tipos de Aprendizado de Máquina existentes hoje estão representados na Figura 1. Técnicas de Aprendizado Supervisionado, por um lado, fazem uso de dados rotulados e da relação entre entrada e saída de dados para aprender durante o treinamento, compreendendo especialmente métodos de regressão e classificação. Técnicas de Aprendizado Não-Supervisionado, por outro lado, apostam na exploração dos padrões dos dados não-rotulados para entender essas relações por conta própria, compreendo métodos como agrupamento. Diferentemente das duas anteriores, o Aprendizado por Reforço aprende por tentativa e erro, melhorando continuamente a cada iteração, através de suas próprias ações e recompensas recebidas.

Hoje é comum encontrar aplicações do Aprendizado de Máquina nos mais variados setores da sociedade, como transporte, finanças e saúde (Gevorkyan *et al.*, 2019). Além disso, a grande disponibilidade e facilidade de acesso à informação, em conjunto com o desenvolvimento de camadas de abstrações para essas técnicas, possibilitou a democratização do uso desse conhecimento, especialmente para profissionais de tecnologia e área correlatas. Estas abstrações estão, em parte, na forma de bibliotecas disponíveis em diferentes linguagens de programação (Gevorkyan *et al.*, 2019), com funções já implementadas e prontas para uso. Algumas bibliotecas podem, inclusive, ser integradas com motores gráficos de jogos, possibilitando o uso de melhores recursos visuais na criação de agentes inteligentes (Juliani *et al.*, 2018).

Ao mesmo tempo que abstrações como essas proporcionam maior facilidade na aplicação

Aprendizado de Máquina ⊆ Inteligência Artificial

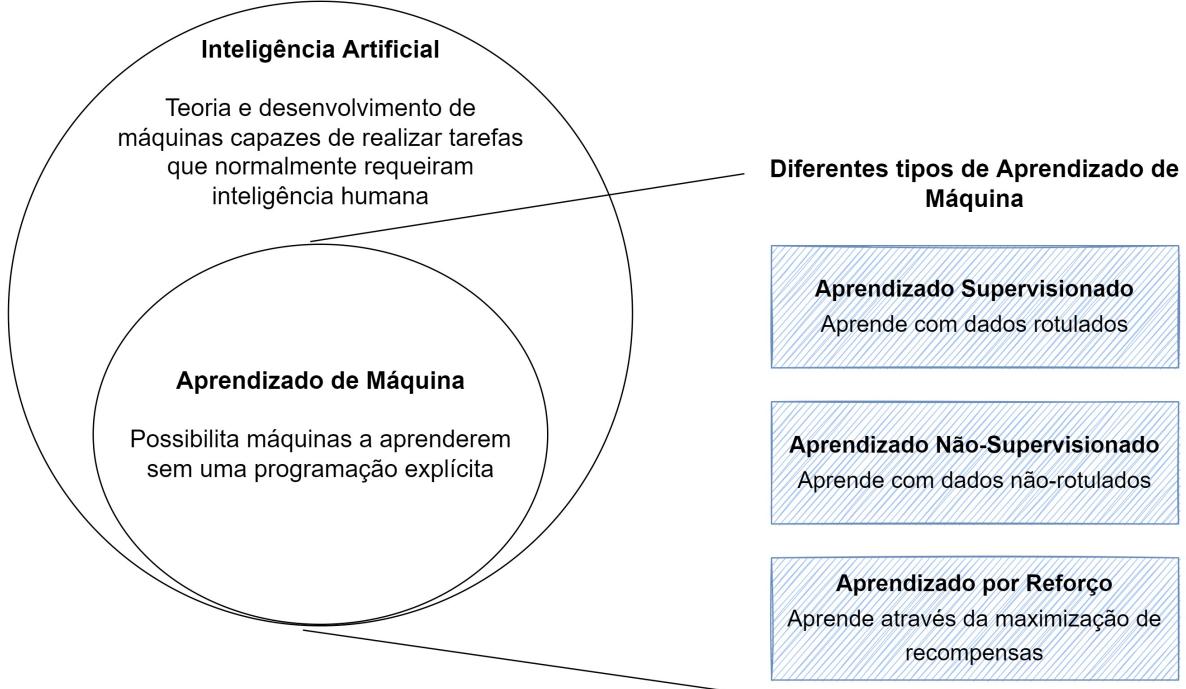


Figura 1: Aprendizado de máquina é uma subárea da Inteligência Artificial e pode ser dividida em três tipos principais: Aprendizagem Supervisionada, Aprendizagem Não Supervisionada e Aprendizagem por Reforço.

de técnicas de aprendizado de máquina, o não conhecimento dos fundamentos pode se tornar uma barreira na implementação de aplicações mais complexas. É importante, desta forma, superar suposições e ultrapassar o entendimento dos algoritmos como uma simples caixa preta, que possui entradas e saídas de conhecimento comum, porém com implementação desconhecida.

De maneira a sanar essas lacunas de conhecimento, existem hoje inúmeras ferramentas que apostam em uma experiência interativa e lúdica. Porém, nem todas as subáreas do campo de Aprendizado de Máquina regozijam de ferramentas robustas e consolidadas, como o *TensorFlow Playground* (Abadi *et al.*, 2016). Muito se concentra em conceitos de Aprendizado Supervisionado e Aprendizado Não-Supervisionado e, apesar do esforço, sobretudo pela academia, em desenvolver ferramentas similares para a área de Aprendizado por Reforço, poucas estão disponíveis facilmente para o público. Ademais, como será discutido na Seção 3, funcionalidades importantes para exploração e visualização não são compartilhadas entre essas diferentes ferramentas, o que pode limitar a experiência do usuário.

Neste projeto, propomos o desenvolvimento de uma ferramenta de visualização de agentes em ambiente de Aprendizado por Reforço interativa e lúdica. Para isso, será utilizado o clássico ambiente *gridworld*, comum na literatura para introdução dos conceitos

básicos, composto por uma grade de células que correspondem aos estados acessíveis ao agente.

A ferramenta desenvolvida neste trabalho proporcionará o uso de algoritmos de diferentes naturezas, tais como Programação Dinâmica (Iteração de Valor e Iteração de Política), apresentados na Seção 2.4, e Diferença Temporal (SARSA e Aprendizagem-Q), apresentados na Seção 2.5. Funcionalidades existentes em ferramentas hoje disponíveis também serão adicionadas e aprimoradas neste projeto. Dentre elas, vale destacar a grande personalização do ambiente de modo a facilitar o entendimento de como diferentes parâmetros dos algoritmos existentes influenciam no aprendizado dos agentes em diferentes ambientes. Mais que isso, propomos a documentação dos conceitos básicos e a disponibilização dos algoritmos em uma linguagem de programação moderna.

2 Conceitos básicos

Nesta seção, apresentamos conceitos básicos de Aprendizado por Reforço considerando como principais referências os trabalhos de Sutton & Barto (2018) e Russell & Norvig (2002) e correlacionando as diferentes definições apresentadas neles.

2.1 Agentes, Ambientes e Racionalidade

Os agentes, em conjunto com o ambiente, são elementos básicos de suma importância para a formalização de uma abordagem computacional de aprendizado por reforço. É importante, portanto, buscarmos nesse primeiro momento uma definição clara de agente, para entendermos como suas interações com o ambiente podem definir dinâmicas de aprendizado e, por consequência, uma noção de racionalidade tal como as dos seres vivos que aprendem com as experiências obtidas em seus meios.

De acordo com Russell & Norvig (2002), podemos definir um agente como uma composição de sensores e atuadores, através dos quais se pode perceber o ambiente ao redor e também agir sobre ele, como apresenta a Figura 2. O mapeamento das percepções obtidas até o momento para alguma ação sobre o ambiente descreve o comportamento deste agente. Na literatura, esse mapeamento é também conhecido como **função do agente**. Desta forma, podemos notar uma dinâmica cíclica entre esses dois elementos: o agente, dado um conjunto de percepções, toma uma ação que altera o ambiente e, por conseguinte, possibilita um novo conjunto de percepções. Em um cenário de aprendizado por reforço, espera-se que o agente aprenda de maneira autônoma tal função, e tome decisões para satisfazer algum objetivo.

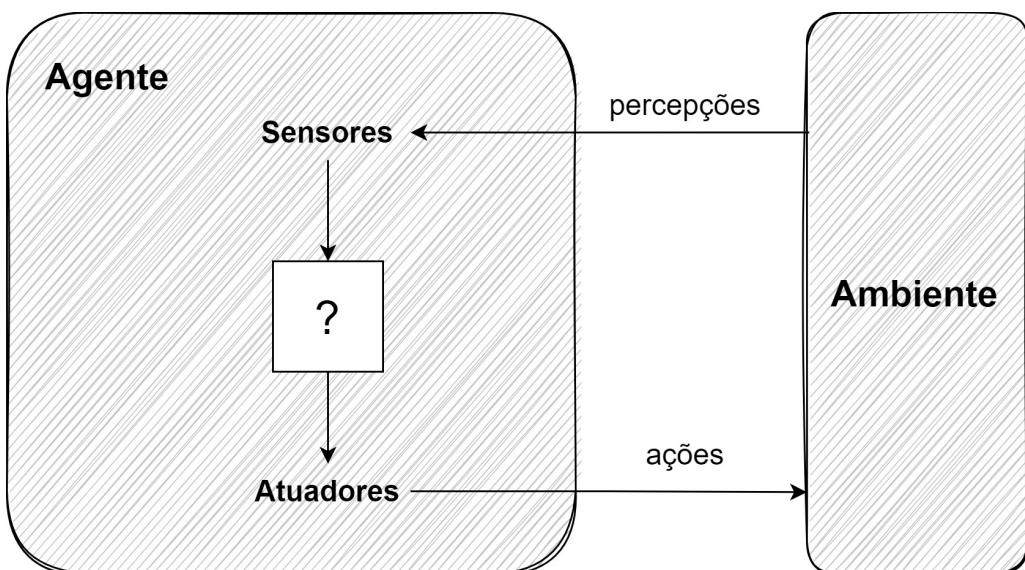


Figura 2: Interação de um agente com o ambiente.

Para ilustrar as definições acima, consideremos um cenário do mundo real: uma pessoa que precisa atravessar a rua. Podemos abstrair esse problema através de um agente humano simples que possui olhos e ouvidos como principais sensores e um par de pernas e de braços como principais atuadores. Neste exemplo, a cada instante o agente pode observar as cores sinalizadas pelo semáforo, o barulho dos veículos ao seu redor ou mesmo sua posição atual, enquanto que pode se movimentar, sinalizar aos motoristas e apertar o botão de travessia do semáforo. E, como apresentado anteriormente, o ambiente se modifica a cada ação tomada: seu movimento implica na atualização de sua posição, motoristas podem reagir à sua sinalização, o estado do semáforo pode ser modificado ao se apertar o botão de travessia.

Neste cenário hipotético, caso o agente humano tivesse uma ação corretamente mapeada para todo estado percebido do ambiente, e agisse, portanto, buscando completar seu objetivo, diríamos que ele é racional. No mundo real, entretanto, nem sempre todas essas entradas são corretas e o agente ou ambiente podem estar dotados de alguma limitação. Da mesma forma, existem diferentes maneiras para aferir o sucesso do objetivo alcançado. É importante, portanto, adotar uma definição concisa para a racionalidade.

Nesse trabalho, utilizaremos a definição de [Russell & Norvig \(2002\)](#), em que um agente é racional se consegue maximizar sua medida de desempenho através das ações que pode executar através da sequência de percepções até o momento e do conhecimento prévio que possui do ambiente. Vale ressaltar, nesta definição, a distinção entre racionalidade e onisciência. O agente humano, por exemplo, quando racional, toma ações para atravessar a rua através de suas percepções até o momento; não se espera que ele consiga prever mudanças inesperadas causadas por fatores externos à sua percepção, como a queda de uma árvore no meio do seu caminho. Em outras palavras, não se espera que o agente seja perfeito. Como colocado anteriormente, isso é impossível no mundo real. A racionalidade, da forma como definida, trata-se da maximização do desempenho esperado.

2.2 Processo de Decisão de Markov

Neste trabalho, destacaremos problemas que exigem tomadas de decisões complexas, em que a preferência do agente de tomar alguma ação depende de uma sequência de decisões. Em um jogo de xadrez, por exemplo, é improvável que consigamos enumerar todos os possíveis estados manualmente, sendo custoso pré-calcular uma tabela de movimentos para cada estado através de um computador. Ambientes estocásticos, que envolvem incertezas, ilustram um cenário semelhante: nas fases de um jogo de *videogame* no estilo de *Super Mario World*, por exemplo, os inimigos em geral têm movimentos bem definidos, repetidos de maneira cíclica e idêntica a cada novo episódio, e a sequência de movimentos

correta sempre poderá ser reutilizada; em contrapartida, se cada inimigo reage com os movimentos do agente, ou se o movimento do agente nem sempre corresponde com ação pretendida, então incertezas devem ser incorporadas nesse processo de tomada de decisão.

Dados esses cenários, de naturezas sequenciais e dotados de incertezas, como pode um agente ser autônomo sem conhecer todos os estados do ambiente? Como pode ele, de maneira escalável, melhorar continuamente o mapeamento de ações para cada estado? Para responder a essas perguntas, vamos recorrer à definição dos **Processos de Decisão de Markov**, também conhecidos *Markov Decision Processes* ou *MDPs* em inglês, que abstraem de maneira simples o problema de aprender através da interação para se atingir um objetivo.

Processos de Decisão de Markov formalizam esse processo de decisão sequencial mencionado ao estender a abstração introduzida na Seção 2.1 através de elementos como recompensa e a definição de estado. Podemos entender o estado como a representação do ambiente observado pelo agente em um instante de tempo t . Nesse novo contexto de interação contínua entre agente e ambiente, as ações tomadas pelo agente em um dado momento influenciam não somente a mudança do ambiente para um novo estado, que gera novas observações, mas também recompensas, imediatas e subsequentes, como medidas do seu desempenho. Essas transições entre estados satisfazem a denominada **propriedade de Markov**, segundo a qual a probabilidade de transição para um estado seguinte depende apenas da ação tomada pelo agente no estado atual, não dependendo, portanto, de estados ou ações anteriores.

A interação entre agente e ambiente em um *MDP* é representada na Figura 3. Nela, ações, estados e recompensas são definidos em função do tempo, de maneira discreta para fins de simplicidade, apesar de ser possível estender os conceitos aqui tratados para caso de tempo contínuo (Doya, 1995).

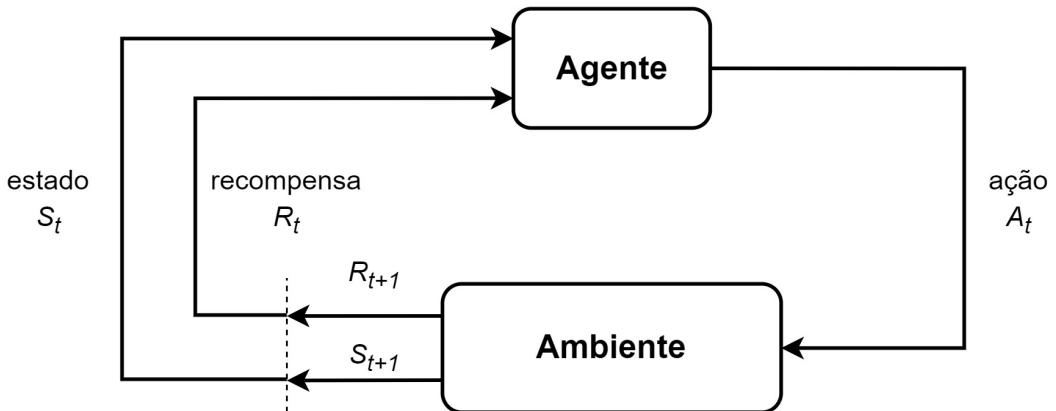


Figura 3: Interação de um agente com o ambiente em um Processo de Decisão de Markov.

Generalizamos, portanto, a dinâmica geral desse sistema através da sequência

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots,$$

em que estado, ação e recompensa em um instante t são representados respectivamente por S_t , A_t e R_t , enquanto S_{t+1} e R_{t+1} correspondem ao estado e recompensa no instante imediatamente seguinte após o agente ter tomado a ação A_t enquanto estava no estado S_t .

Vale ressaltar que esta sequência apresenta um número ilimitado de iterações. Isso, entretanto, nem sempre é verdade para o problema retratado. Um robô aspirador que recebe recompensas positivas somente a cada quadrante limpo, por exemplo, talvez não precise funcionar continuamente se o ambiente não for constantemente sujo, bastando passar por todo o ambiente para concluir a faxina diária. Muitas vezes, a maneira como definimos o objetivo do agente, ou a própria natureza do problema, proporciona naturalmente uma noção de fim. Para esses problemas, podemos, portanto, definir a sequência finita

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_{T-1}, A_{T-1}, R_T, S_T.$$

Esta sequência, chamada por [Sutton & Barto \(2018\)](#) de episódica, termina em um estado especial, representado por S_T , onde T é último instante de toda essa interação. É importante notar também que consideramos neste trabalho *MDPs* finitos, isto é, que possuem um conjunto finito de estados S , de ações A e de recompensas R .

2.2.1 Estocasticidade

Em Processos de Decisão de Markov, as dinâmicas do ambiente são descritas por uma distribuição de probabilidade discreta. Na formalização utilizada por [Sutton & Barto \(2018\)](#), mais genérica, o estado e a recompensa obtida dependem exclusivamente da ação tomada em um estado do instante anterior:

$$p(s', r | s, a) \doteq Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}. \quad (1)$$

Essa modelagem, porém, pode variar. Como observaremos na Seção 2.3, em um ambiente de *gridworld*, composto por uma grade de células que correspondem aos estados, onde as recompensas são providas unicamente através do estado atual do agente, não dependendo da ação, podemos calcular as probabilidades das transições entre estados:

$$p(s' | s, a) = \sum_{r \in R} p(s', r | s, a). \quad (2)$$

Um exemplo prático do uso da Equação (2) é mostrado na Figura 4. Nele, um robô aspirador, inicialmente em um estado S_0 , pode se movimentar para quatro outros estados adjacentes S_1 , S_2 , S_3 e S_4 . Seu movimento, entretanto, é estocástico e dotado de incertezas, o que o faz desviar para cada lado com 10% de chance e seguir na direção correta em somente 80% dos casos.

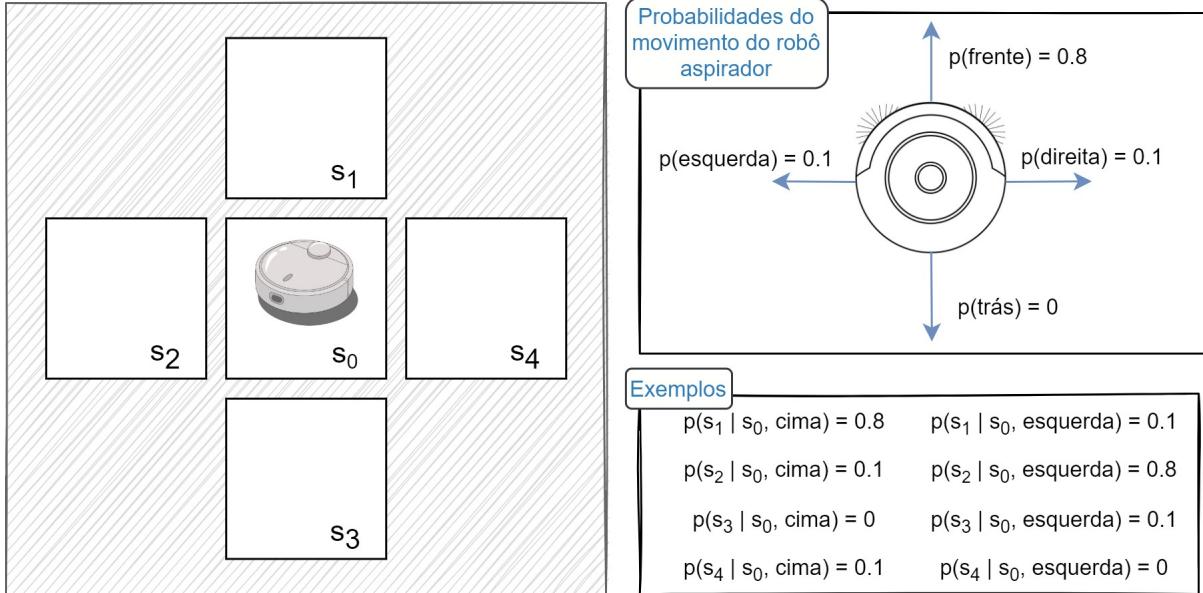


Figura 4: Probabilidades das transições entre estados de um robô aspirador que se movimenta de maneira estocástica para frente, trás, direita e esquerda.

2.2.2 Recompensas

Dada a definição genérica de probabilidade pela Equação (1), podemos computar a recompensa para qualquer ação tomada pelo agente em um dado estado através do valor esperado da recompensa, considerando todos os estados futuros possíveis:

$$r(s, a) = \mathbb{E} [R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r | s, a). \quad (3)$$

As recompensas, como discutido anteriormente, servem como estímulo ao agente para o processo de aprendizado, similar a uma medida de desempenho. Este estímulo pode ser dado de diferentes formas, de maneira positiva ou negativa, sendo considerado para entender quão bons são cada um dos estados para se atingir um objetivo.

Cada problema pode requerer um sistema de recompensas diferente: robôs aspiradores, por exemplo, podem ser recompensados por cada pedaço limpo em cada instante, enquanto que um agente que precisa sair de um labirinto pode ser recompensado somente ao final. Nesses dois casos, pode-se também penalizar, com recompensas negativas, como estímulo para o agente terminar a tarefa o quanto antes.

Espera-se que o agente maximize as recompensas recebidas, tanto as imediatas quanto as de longo prazo. Para isso, podemos atribuir um valor de retorno para as sequências de estados, partindo de um instante t , e considerando todas as recompensas obtidas até o fim do episódio. Esse valor é denotado por G_t na equação

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T. \quad (4)$$

Para problemas de natureza não-episódica ou cujos episódios se estendem por um período muito longo, é interessante que consigamos incorporar a preferência de um agente por recompensas mais imediatas, de maneira que recompensas futuras sejam consideradas menos relevantes para o processo de aprendizado. Podemos, portanto, generalizar a equação anterior ao adicionar um fator de desconto γ , que varia entre 0 e 1:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \quad (5)$$

Na Equação (5), quando $\gamma = 0$, o agente só aprende com recompensas imediatas. Quando $\gamma = 1$, o comportamento é o mesmo da Equação (4).

2.2.3 Políticas e Funções-valores

Como relatado anteriormente, todo agente é dotado de uma função de agente, que mapeia as diferentes percepções obtidas do ambiente para uma consequente ação. Entendemos também, nas seções anteriores, como as essas diferentes percepções podem ser abstraídas a um estado específico do ambiente. Na literatura, essas funções do agente são incorporadas por políticas, denotadas por π . Representamos, desta forma, a ação recomendada para um agente em um estado s através de $\pi(s)$.

Podemos utilizar as sequências de recompensas recebidas pelo agente em suas diversas interações com o ambiente, e seus valores de retorno definidos através da Equação (5), para estimar o quão relevante é cada estado para o alcance do objetivo final. Conhecida como **utilidade** na literatura, representamos esta estimativa para um estado seguindo uma dada política π através de $v_\pi(s)$, tal como define a equação

$$v_\pi(s) \doteq \mathbb{E}_\pi [G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]. \quad (6)$$

Da mesma forma que definimos o quão útil para o agente é estar em um estado para atingir seu objetivo, podemos definir o quão propícia é a tomada de uma ação específica em um estado seguindo uma dada política π . Essa estimativa é conhecida

como **qualidade**, definida através da equação

$$q_\pi(s, a) \doteq \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]. \quad (7)$$

Qualidade e utilidade são termos amplamente utilizadas na literatura, como é feito por [Russell & Norvig \(2002\)](#), mas podemos também denominá-los, de uma maneira mais geral, por **funções-valores**, como faz [Sutton & Barto \(2018\)](#).

Podemos, ainda, tentar estender a definição de valor de retorno utilizada em ambas as funções-valores, de maneira a relacionar as recompensas de diferentes instantes da interação do agente com o ambiente, como faz a equação

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned} \quad (8)$$

E como resultado, obtemos a **Equação de Bellman**, que possui grande relevância nos algoritmos explorados neste trabalho por expressar uma relação recursiva entre a utilidade de dois estados sucessivos:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi [G_t \mid S_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_{s',r} p(s', r \mid s, a) [r + \gamma v_\pi(s')]. \end{aligned} \quad (9)$$

2.2.4 Optimalidade

Como vimos, as funções-valores consideram a política atual do agente em suas formulações para computar a recompensa esperada obtida em cada estado. Se tal recompensa esperada obtida com uma política π_1 é maior ou igual à obtida com outra política π_2 para todos os estados, então pode-se naturalmente dizer que π_1 é melhor ou igual que π_2 , pois com ela o agente obtém recompensas similares ou maiores em suas interações com o ambiente. Como o mesmo é válido para o caminho contrário, podemos definir:

Proposição 1 $\pi_1 \geq \pi_2 \iff v_{\pi_1}(s) \geq v_{\pi_2}(s) \quad \forall s \in S.$

Em decorrência da Proposição 1, e atentando-se ao fato de que haverá sempre pelo menos uma política melhor ou igual que qualquer outra, podemos definir tal política como ótima, ou simplesmente π_* . É válido notar que esta é uma política ótima por conseguir alcançar o objetivo de maximizar as recompensas obtidas pelo agente, representadas pelas utilidades, neste caso também ótimas, e denotadas por v_* .

Como observaremos nas próximas seções, não somente as utilidades são ditadas pela política do agente, como também podem servir de parâmetro para geração de novas políticas. Em um processo de aprendizado por reforço, espera-se a obtenção de políticas cada vez melhores e próximas de uma política ótima. Essa relação mútua serve de base para os algoritmos que serão abordados neste trabalho. Porém, antes de avançar neste tópico, precisamos entender algumas relações de otimalidade geradas a partir do que já foi apresentado.

Considerando que um agente com uma política ótima sempre toma a ação que maximiza sua recompensa, é sensato afirmar que sua utilidade em algum estado deva ser equivalente à qualidade de tomar a melhor ação naquele mesmo estado. Representamos essa relação na equação

$$\begin{aligned} v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}_{\pi_*} [G_t \mid S_t = s, A_t = a]. \end{aligned} \tag{10}$$

Utilizando (8) em (10) e considerando ainda a definição de utilidade dada pela Equação (6), mas em um contexto de otimalidade, obtemos a equação

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}_{\pi_*} [R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E} [R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]. \end{aligned} \tag{11}$$

Por fim, chegamos na **Equação de otimalidade de Bellman**:

$$v_*(s) = \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')]. \tag{12}$$

Além de apresentar a dependência entre utilidades de diferentes estados, como faz a Equação de Bellman inicialmente deduzida, esta nova equação possibilita uma maneira

de encontrar uma solução ótima, por ser na verdade um sistema de equações, um para cada estado.

Da mesma forma realizada para a utilidade, podemos chegar em um resultado para a qualidade:

$$\begin{aligned} q_*(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\ &= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]. \end{aligned} \quad (13)$$

Nessa equação, entretanto, em vez de atualizar o valor do estado atual conforme o estado subsequente de maior função-valor (dada pela utilidade), consideramos a maior função-valor de todos os estados subsequentes (dada pela qualidade). Isso pode ser entendido através da Figura 5, que apresenta os diagramas referentes à atualização da utilidade e qualidade de um estado através dos estados sucessores. Nesse diagrama, círculos brancos representam estados, enquanto círculos pretos representam pares estado-ação. Como se pode observar nele, ao focar somente na utilidade do estado, consideramos todas as ações que o agente pode tomar e, para cada uma dessas ações, todos os possíveis estados subsequentes, enquanto que essa ação já é conhecida ao focar na qualidade, bastando olhar para a qualidade dos estados alcançáveis através desta ação.

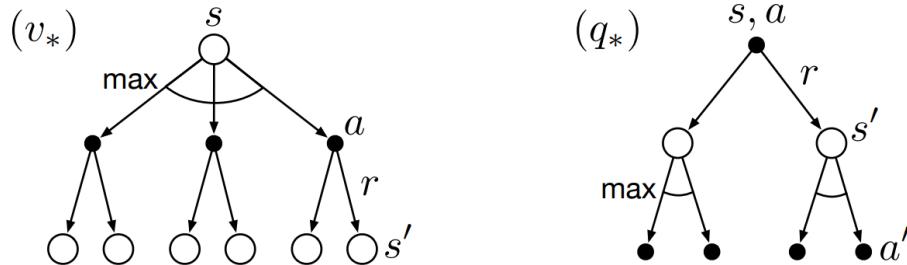


Figura 5: Diagramas de atualização de utilidade (à esquerda) e qualidade (à direita) durante aprendizado por reforço. Círculos brancos representam estados e círculos pretos representam pares estado-ação.

2.3 Gridworld

O ambiente é um importante elemento que interage com o agente. Neste trabalho, aplicaremos os conceitos no *gridworld*. Portanto, é importante entendermos as dinâmicas envolvidas, bem como adequações das fórmulas vistas até o momento para esse contexto.

O *gridworld* é um ambiente composto por uma grade de células que correspondem aos

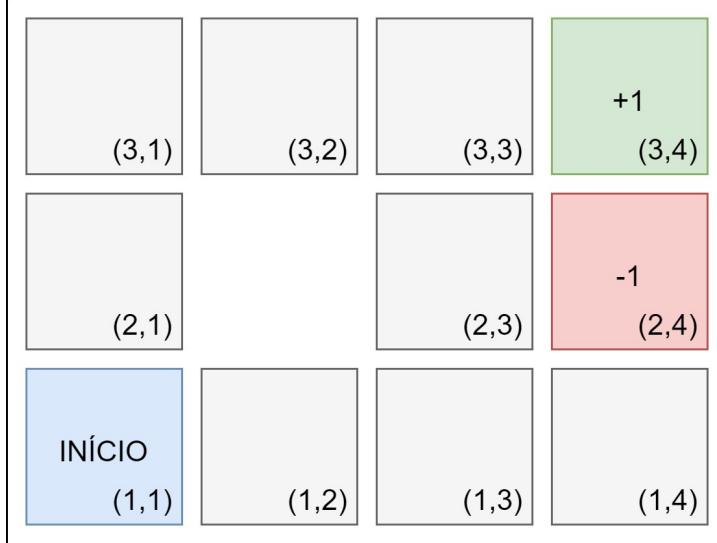


Figura 6: Gridworld clássico, composto por um estado inicial (em azul), um estado terminal desejável (em verde) e um estado terminal que deve ser evitado (em vermelho).

estados por onde o agente pode se movimentar. Em sua forma mais clássica utilizada na literatura, sua configuração consiste em um mundo de tamanho 3x4, como apresenta a Figura 6. Neste mundo, o agente pode realizar movimentos nas quatro direções cardinais, podendo avançar, portanto, somente para células adjacentes. Além disso, assumimos que o agente não pode sair do mundo, portanto, quando um movimento é dado em uma direção não permitida, o agente permanece no mesmo estado.

Uma peculiaridade referente a esse ambiente é a maneira como o agente recebe recompensas pelo seu comportamento. No *gridworld*, cada estado possui uma recompensa fixa, e o agente recebe a recompensa referente ao estado em que estiver após ter tomado alguma ação. Na configuração da Figura 6, cada estado garante ao agente uma recompensa negativa de $-0,04$, com exceção do estado terminal que deve ser alcançado, representado na cor verde na posição (3,4), com recompensa +1, e do estado terminal que deve ser evitado, representado na cor vermelha na posição (2, 4), de recompensa -1. Nesta configuração, há também um estado inicial, representado na cor azul na posição (1, 1).

Diferentemente do que apresentamos até o momento, a recompensa nesse ambiente específico depende somente do estado, não havendo nenhuma relação com a ação tomada. Além disso, para cada ação a tomada em um estado s , há uma única recompensa r_s possível recebida pelo agente ao se mover para um estado sucessivo s' . Logo, para todos os outros $r \in R$, podemos aferir que $p(s', r | s, a) = 0$. Ao aplicar isso na Equação de Optimalidade de Bellman (12), e considerar também que $p(s', r | s, a) = p(s' | s, a)$ neste contexto, chegamos na equação

$$\begin{aligned}
v_*(s) &= \max_a \sum_{s',r} p(s',r | s,a) [r + \gamma v_*(s')] \\
&= \max_a \sum_{s'} \sum_r p(s',r | s,a) [r + \gamma v_*(s')] \\
&= \max_a \left[\sum_{s'} p(s',r_s | s,a) [r_s + \gamma v_*(s')] + \sum_{s'} \sum_{r \neq r_s} \cancel{p(s',r | s,a)}^0 [r + \gamma v_*(s')] \right] \\
&= \max_a \sum_{s'} p(s' | s,a) [r_s + \gamma v_*(s')] .
\end{aligned} \tag{14}$$

Por fim, podemos aplicar a propriedade distributiva da multiplicação dos termos na equação anterior, e chegar em uma adaptação da Equação de Optimalidade de Bellman para o contexto de *gridworld*:

$$\begin{aligned}
v_*(s) &= \max_a \sum_{s'} p(s' | s,a) [r_s + \gamma v_*(s')] \\
&= \max_a \sum_{s'} p(s' | s,a) r_s + \max_a \sum_{s'} p(s' | s,a) \gamma v_*(s') \\
&= r_s \max_a \sum_{s'} \cancel{p(s' | s,a)}^1 + \gamma \max_a \sum_{s'} p(s' | s,a) v_*(s') \\
&= r_s + \gamma \max_a \sum_{s'} p(s' | s,a) v_*(s').
\end{aligned} \tag{15}$$

De maneira similar, porém mais simples, podemos adaptar a Equação simplificada de Bellman (9) no contexto do *gridworld*:

$$\begin{aligned}
v_\pi(s) &= \sum_{s',r} p(s',r | s,a) [r + \gamma v_\pi(s')] \\
&= \sum_{s'} \sum_r p(s',r | s,a) [r + \gamma v_\pi(s')] \\
&= \left[\sum_{s'} p(s',r_s | s,a) [r_s + \gamma v_\pi(s')] + \sum_{s'} \sum_{r \neq r_s} \cancel{p(s',r | s,a)}^0 [r + \gamma v_\pi(s')] \right] \\
&= \sum_{s'} p(s' | s,a) [r_s + \gamma v_\pi(s')] \\
&= \sum_{s'} \cancel{p(s' | s,a)}^1 r_{s'} + \gamma \sum_{s'} p(s' | s,a) v_\pi(s') \\
&= r_{s'} + \gamma \sum_{s'} p(s' | s,a) v_\pi(s').
\end{aligned} \tag{16}$$

2.4 Algoritmos de Programação Dinâmica

Na Seção 2.2.4, introduzimos as equações de otimalidade de Bellman e discutimos como elas definem um sistema de equações, uma para cada estado, relacionadas entre si. Se conhecemos as dinâmicas do ambiente, dadas pela probabilidade de o agente mudar de estado por uma ação, então bastaria encontrar a solução com algum método de equações não-lineares. Neste capítulo, entretanto, exploramos uma solução computacional mais simples, porém custosa, que surge da recursão, e como aplicar programação dinâmica para melhor desempenho, através do apoio de uma tabela de valores pré-computados.

2.4.1 Algoritmo de Iteração de Política

Uma primeira tentativa de encontrar a política ótima pode ser extraída da relação mútua entre função-valor e política, em conjunto com as equações que vimos até o momento. Como sabemos, definimos utilidades e qualidades a partir da política atual do agente. Entretanto, como já foi antecipado, podemos gerar novas políticas com as utilidades obtidas. Se entendermos como gerar políticas cada vez melhores, a melhoria contínua fruto da repetição dessas duas etapas tenderá para a otimalidade.

A primeira etapa deste algoritmo é denominada **Avaliação de Política** e consiste em computar funções-valores a partir de políticas arbitrárias considerando a relação recursiva entre utilidade de dois estados sucessivos, existente para todo estado do *MDP*, definida pela Equação de Bellman. Neste projeto vamos usar a Equação (16), adaptada para o contexto do *gridworld*.

Apesar de existirem métodos clássicos de álgebra linear para resolver esse sistema de equações que podem ser resolvidos numa complexidade de tempo $O(n^3)$, onde n é o número de estados, seu custo computacional pode ser proibitivo para muitos estados (Russell & Norvig, 2002). Desta forma, a abordagem geral para a avaliação de política se dá por sucessivas aproximações, a partir de uma pequena modificação da Equação de Bellman para definir uma regra de atualização:

$$v_{k+1}(s) = r_{s'} + \gamma \sum_{s'} p(s' | s, a) v_k(s'). \quad (17)$$

Desta forma, partindo de uma aproximação inicial v_0 escolhida de maneira arbitrária e diante da atualização de todos os valores, seguindo o comportamento do agente com sua política π , podemos computar v_π através da convergência da sequência de funções-valores v_0, v_1, v_2, \dots calculadas. O Algoritmo 1 implementa essa ideia aplicando programação dinâmica para cada sucessiva aproximação (Sutton & Barto, 2018), resolvendo, portanto, um problema de recursão com o apoio de uma tabela e de valores pré-computados.

Algoritmo 1: Avaliação de Política

Entrada: MDP , composto por um conjunto de estados S , de ações A e de recompensas R , modelo de transição $p(s' | s, a) \forall s \in S, a \in A$
Função-valor V
Fator de desconto γ
Fator de acurácia da estimativa θ

Saída: Função-valor V

1 **início**

```
2   repita
3      $\Delta \leftarrow 0;$ 
4     para cada  $s \in S$  faz
5        $v \leftarrow V(s);$ 
6        $V(s) \leftarrow r_s + \gamma \sum_{s'} p(s' | s, a) V(s');$ 
7        $\Delta \leftarrow \max(\Delta, |v - V(s)|);$ 
8     fim
9   até  $\Delta < \theta;$ 
10  retorna  $V$ 
11 fim
```

A partir das funções-valores computadas, queremos encontrar políticas melhores. Em outras palavras, para cada estado, precisamos avaliar a necessidade de mudar a política de maneira ao agente tomar uma nova ação $a \neq \pi(s)$. Essa etapa é conhecida como **Aperfeiçoamento de Política**, extraída naturalmente da Equação de Optimalidade de Bellman. Novamente, para este trabalho vamos usar a Equação (15) adaptada para o contexto do *gridworld*. Neste caso, entretanto, estamos interessados na ação que maximiza a utilidade, em vez do valor.

A etapa de Aperfeiçoamento de Política é executada logo após a Avaliação de Política, e exige entender se houve alguma estabilidade da política, isto é, se nenhuma ação dela foi modificada, para interromper a execução, devido à convergência, ou voltar ao primeiro passo do ciclo. As duas etapas são mostradas no Algoritmo 2.

Algoritmo 2: Iteração de Política

Entrada: MDP , composto por um conjunto de estados S , de ações A e de recompensas R , modelo de transição $p(s' | s, a) \forall s \in S, a \in A$
Fator de desconto γ

Saída: Política π
Função-valor V

1 **início**

2 Inicialize $V(s) = 0 \forall s \in S$;
3 Inicialize $\pi(s) \in A$ arbitrariamente $\forall s \in S$;
4 **repita**
 5 $V \leftarrow \text{AVALIACAO_DE_POLITICA}(mdp, V, \gamma)$;
 6 $politica_estavel \leftarrow verdadeiro$;
 7 **para** cada $s \in S$ **faz**
 8 $acao_anterior \leftarrow \pi(s)$;
 9 $\pi(s) \leftarrow \text{argmax}_a \sum_{s'} p(s' | s, a) V(s')$;
 10 **se** $acao_anterior \neq \pi(s)$ **então**
 11 $politica_estavel \leftarrow \text{falso}$;
 12 **fim**
 13 **fim**
 14 **até** $politica_estavel = \text{falso}$;
 15 **retorna** π, V

16 **fim**

Em suma, pode-se resumir este algoritmo através de uma sequência cíclica do melhoramento das funções-valores e das políticas, até se perceber uma convergência:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*},$$

onde \xrightarrow{E} denota a Avaliação de Política e \xrightarrow{I} denota o Aperfeiçoamento de Política.

2.4.2 Algoritmo de Iteração de Valor

No algoritmo de Iteração de Política, utilizamos a Equação de Otimalidade de Bellman somente para avaliar a política, de maneira a estimar a utilidade para então obter melhores estimativas da política. Mas, essa própria equação poderia ser usada como regra de atualização em um único ciclo de iterações, visto que ela incorpora a Equação de Bellman original e combina a melhoria de política com a avaliação de política:

$$v_*(s) = r_{s'} + \gamma \max_a \sum_{s'} p(s' | s, a) v_*(s'). \quad (18)$$

É valido notar, neste caso, que a avaliação de política ocorre de maneira truncada, não sendo uma convergência das utilidades para valores que não influenciem na mudança da política. O Algoritmo 3 implementa essa ideia similarmente aos algoritmos já vistos, por sucessivas aproximações, e retorna a função-valor.

Algoritmo 3: Iteração de Valor

Entrada: MDP , composto por um conjunto de estados S , de ações A e de recompensas R , modelo de transição $p(s' | s, a) \forall s \in S, a \in A$
Fator de desconto γ

Saída: Função-valor V

1 **início**

2 Incialize $V(s) = 0 \forall s \in S$;

3 **repita**

4 $\Delta \leftarrow 0$;

5 **para** cada $s \in S$ **faca**

6 $v \leftarrow V(s)$;

7 $V(s) \leftarrow r_s + \gamma \max_a \sum_{s'} p(s' | s, a) V(s')$;

8 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;

9 **fim**

10 **até** $\Delta < \theta$ (*acurácia da estimativa*);

11 **retorna** π, V

12 **fim**

A política pode ser definida após convergência da função-valor para um estado ótimo, de maneira bem similar ao que é feito no algoritmo de Iteração Política, atualizando a ação para cada estado através da seguinte regra:

$$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s'} p(s' | s, a) V(s') \quad (19)$$

2.5 Algoritmos de Diferença Temporal

Até esse momento, trabalhamos com a suposição de um modelo perfeito, isto é, de que temos total conhecimento das dinâmicas do ambiente, como as probabilidades em cada mudança de estado. Utilizamos esse conhecimento, por exemplo, nos algoritmos de Iteração de Valor e de Iteração de Política. Entretanto, nem sempre conseguimos ter acesso a

esse tipo de informação de maneira precisa em problemas de maior complexidade, especialmente quando lidamos com grande número de estados, como um jogo de xadrez. Na impossibilidade de usar esse conhecimento prévio para treinar a função do agente, surge a estratégia de fazer com que o agente aprenda essa função de maneira autônoma através de suas próprias experiências.

Essa estratégia serve como estrutura para os algoritmos de Diferença Temporal, que tomam um espaço muito importante na Área de Aprendizado por Reforço.

2.5.1 Aprendizagem por experiência

Existem diferentes formas de aprender através da própria experiência, sendo a mais básica a **Estimativa de Utilidade Direta**. Nesse método, o agente executa uma série de episódios, do estado inicial até um dos terminais, que garantem amostras das recompensas obtidas pelo agente. As funções-valores, desta forma, podem ser calculadas de maneira simples considerando o valor médio das recompensas recebidas de um dado estado até o estado terminal. Como exemplo, consideremos o seguinte episódio:

$$(1, 1)_{-0.04} \rightarrow (1, 2)_{-0.04} \rightarrow (1, 3)_{-0.04} \rightarrow (1, 2)_{-0.04} \rightarrow (1, 3)_{-0.04} \rightarrow \\ (2, 3)_{-0.04} \rightarrow (3, 3)_{-0.04} \rightarrow (4, 3)_{+1}.$$

A sequência de estados vistados neste episódio é apresentada de maneira gráfica na Figura 7. Neste exemplo, o total de recompensas recebidas começando pelo estado inicial $(1, 1)$ é de 0,72. É possível notar que o estado $(1, 2)$ é visitado duas vezes, tendo um total de recompensas de 0,76 na primeira passada, e de 0,84 na segunda passada. Para este caso, portanto, consideramos o valor médio, de 0,8. Usamos, também, o valor médio entre os totais acumulados de cada episódio.

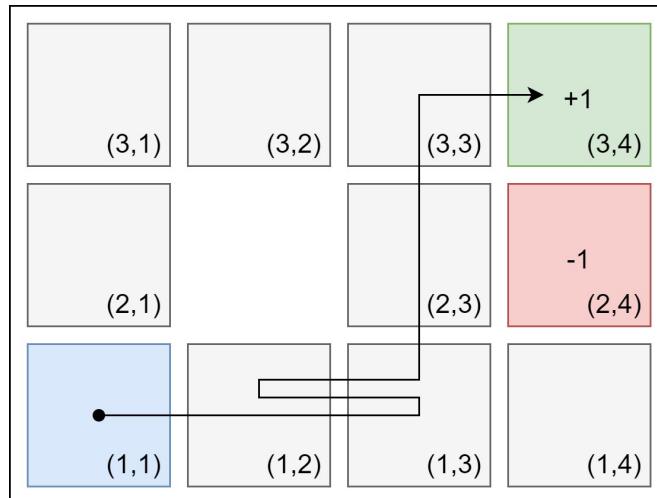


Figura 7: Exemplo de um episódio vivenciado pelo ambiente no gridworld.

É importante notar que, para simplificar o problema, este método supõe de maneira equívoca que as funções-valores dos estados são independentes. Mesmo partindo disso, essa é uma estratégia que costuma funcionar para muitos casos. Podemos, entretanto, utilizar uma estratégia similar considerando as propriedades de Markov, através de algoritmos de Diferença Temporal, discutidos nas próximas subsecções.

2.5.2 SARSA

No algoritmo SARSA, a estimativa é feita também através das experiências obtidas pelo agente nos diferentes episódios, porém de uma maneira mais precisa, por estimar não somente a recompensa recebida por passar por algum estado, mas também considerar a recompensa esperada do próximo estado, através da equação:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] . \quad (20)$$

É importante notar a adição de uma taxa de aprendizado α , que representa a magnitude do passo que é dado em direção à solução, isto é, o quanto se deve aceitar a recompensa esperada do próximo estado.

O Algoritmo 4 implementa não somente a ideia de exploração do agente, mas também a correta atualização das funções-valores. Esse método pode ser classificado como *on-policy*, por tentar melhorar a mesma política utilizada para tomar as decisões em cada episódio.

Algoritmo 4: SARSA

Entrada: MDP , composto por um conjunto de estados S , de ações A e de recompensas R , modelo de transição $p(s' | s, a) \forall s \in S, a \in A$
 Fator de desconto γ
 Taxa de aprendizado α

Saída: Função-valor Q

```

1 início
2   Inicialize  $Q(s, a) = 0 \forall s \in S, a \in A;$ 
3   para cada episódio faça
4      $s \leftarrow$  estado inicial do  $mdp$ ;
5      $a \leftarrow$  escolha uma ação a partir de  $s$  usando política derivada de  $Q$ ;
6     para cada passo do episódio faça
7       se  $s$  é terminal então
8          $Q[s] = r;$ 
9         retorna
10         $s', r \leftarrow$  tome a ação  $a$ ;
11         $a' \leftarrow$  escolha uma ação a partir de  $s'$  usando política derivada de  $Q$ ;
12         $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$ 
13         $s \leftarrow s'$ ;
14         $a \leftarrow a'$ ;
15      fim
16    fim
17    retorna  $Q$ 
18 fim

```

2.5.3 Aprendizagem-Q

O algoritmo de Aprendizagem-Q compartilha as mesmas ideias do SARSA, e possui uma estrutura geral muito similar, porém parte para uma estratégia *off-policy*, isto é, tenta avaliar ou aperfeiçoar uma política diferente daquela utilizada para gerar as observações. Isto pode ser notado através da equação:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \quad (21)$$

Neste caso, a função Q , aprimorada a cada sucessiva iteração, se aproxima de um valor ótimo considerando a ação que maximiza a recompensa esperada do próximo estado. Essa diferença pode ser melhor compreendida ao compararmos os algoritmos 4 e 5.

Algoritmo 5: Aprendizagem-Q

Entrada: MDP , composto por um conjunto de estados S , de ações A e de recompensas R , modelo de transição $p(s' | s, a) \forall s \in S, a \in A$
Fator de desconto γ
Taxa de aprendizado α

Saída: Função-valor Q

1 **início**

2 Inicialize $Q(s, a) = 0 \forall s \in S, a \in A$;
3 **para** cada episódio **faça**
4 $s \leftarrow$ estado inicial do mdp ;
5 **para** cada passo do episódio **faça**
6 **se** s é terminal **então**
7 $Q[s] = r$;
8 **retorna**
9 $a \leftarrow$ escolha uma ação a partir de s usando política derivada de Q ;
10 $s', r \leftarrow$ tome a ação a ;
11 $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_a Q(s', a') - Q(s, a)]$
12 $s \leftarrow s'$
13 **fim**
14 **fim**
15 **retorna** Q
16 **fim**

3 Trabalhos correlatos

As ferramentas existentes hoje para visualização de Aprendizado por Reforço apostam na mesma estratégia didática adotada pela literatura ao apresentar conceitos básicos e introduzir o funcionamento dos algoritmos através do uso do *gridworld*. Como apresentado na Seção 2, o *gridworld* consiste em um ambiente composto por uma grade de células que correspondem aos estados por onde o agente pode se movimentar e que podem gerar recompensas positivas ou negativas a ele quando alcançados. Somam-se a esse conjunto de informações, a percepção do agente em relação ao mundo (de como os estados o ajudam a alcançar seus objetivos e das decisões que deve tomar a cada célula alcançada) e informações inerentes à execução do algoritmo, como os parâmetros utilizados e a modificação do comportamento do agente a cada iteração.

Entre os esforços realizados pela comunidade científica, Karpathy (2015) propõe o *REINFORCEjs*, uma biblioteca de Aprendizado por Reforço para a linguagem JavaScript que implementa diferentes algoritmos, dentre eles os de Programação Dinâmica (Iteração de Valor e Iteração de Política) e de Diferença Temporal (SARSA e Aprendizagem-Q), todos com demonstrações interativas disponíveis na web. As demonstrações contam, inclusive, com algoritmos cuja natureza está além do escopo deste trabalho, como aprendizado profundo. Elemento essencial da demonstração, a Figura 8 mostra como o usuário pode dar início ao algoritmo e comandar o tempo de execução através de botões. Em uma caixa de texto, na forma variáveis, são instanciados os parâmetros utilizados durante o aprendizado, como a taxa de aprendizado α , o fator de desconto γ , ou mesmo o próprio tipo de algoritmo. Estes parâmetros podem ser modificados pelo usuário.

```
// agent parameter spec to play with (this gets eval()'d on Agent reset)
var spec = {}
spec.update = 'qlearn'; // 'qlearn' or 'sarsa'
spec.gamma = 0.9; // discount factor, [0, 1]
spec.epsilon = 0.2; // initial epsilon for epsilon-greedy policy, [0, 1]
spec.alpha = 0.1; // value function learning rate
spec.lambda = 0; // eligibility trace decay, [0,1]. 0 = no eligibility traces
spec.replacing_traces = true; // use replacing or accumulating traces
spec.planN = 50; // number of planning steps per iteration. 0 = no planning
spec.smooth_policy_update = true; // non-standard, updates policy smoothly to follow max_a Q
spec.beta = 0.1; // learning rate for smooth policy update
```

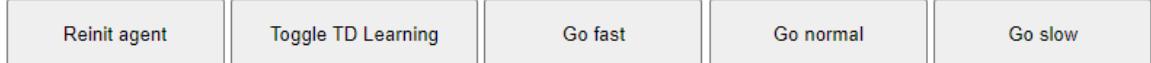


Figura 8: Personalização de parâmetros na ferramenta *REINFORCEjs*.

Durante a execução do algoritmo selecionado, o *gridworld* é constantemente atualizado com informações sobre a utilidade e política atual de cada estado, e movimentação do

agente, indicado por um círculo amarelo. Como apresenta a Figura 9, é possível observar a transição de política dos estados através do redimensionamento do tamanho das setas, que podem apontar para os quatro pontos cardinais. Os estados são coloridos de acordo com a utilidade e variam do vermelho, no pior caso, ao verde, no melhor caso, o que auxilia na visualização do ambiente como um todo. Nessa demonstração, o usuário pode ainda modificar o fator ϵ de exploração (que reflete na probabilidade de o agente tomar ações aleatórias e, portanto, explorar novos estados) e a recompensa de cada estado para um valor entre -5 e 5.

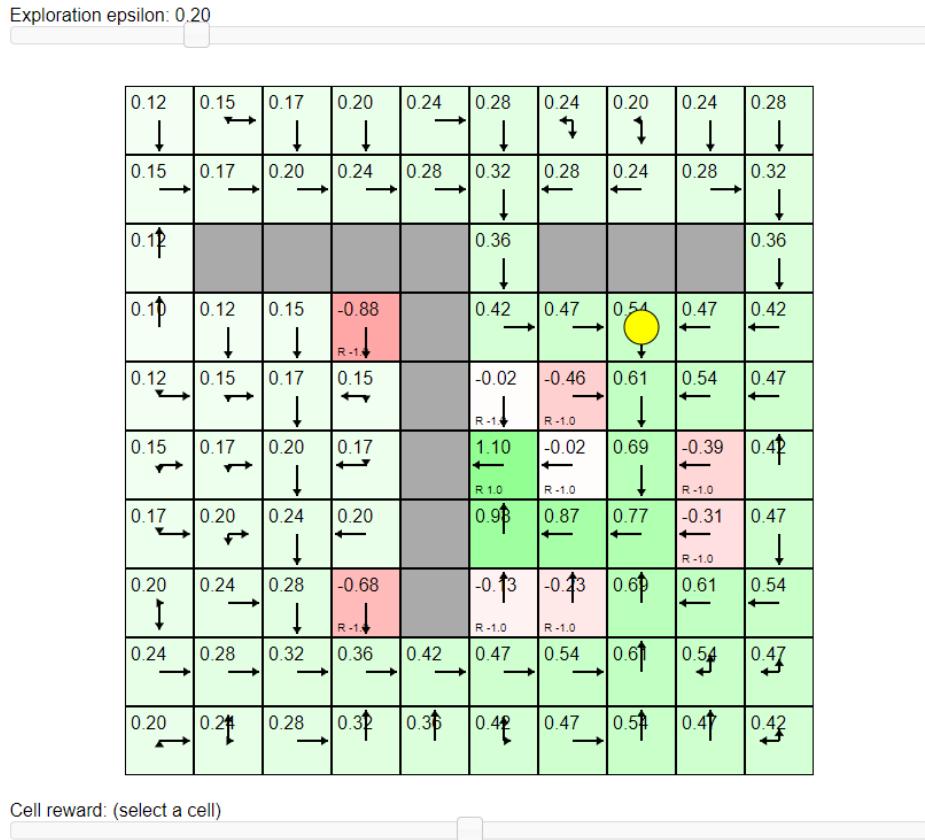


Figura 9: Visualização do gridworld na ferramenta REINFORCEjs.

Apesar de o trabalho de Karpathy (2015) introduzir uma demonstração com uma variedade de elementos de visualização e interação, existem algumas limitações: não é possível modificar a estrutura do *gridworld*, adicionando ou removendo estados, ou definir o número de iterações.

Essas funcionalidades são implementadas no trabalho de Kumar (2020), de nome *GridWorld RL*, que diferentemente do trabalho anterior, é exclusivamente uma ferramenta de visualização. Sua interface principal é apresentada na Figura 10 e dispõe da customização de dois parâmetros: o fator de desconto γ e ruído, que influencia no determinismo das ações dos agentes. É possível modificar a utilidade inicial dos estados (de maneira coletiva, não individualmente) e propor uma recompensa a cada movimentação

do agente, além daquela obtida a cada estado. O *grid*, por padrão, é o mesmo utilizado na literatura para introdução dos conceitos (apresentado na Seção 2.3), mas pode ser modificado com a adição ou remoção de células e personalização de recompensas.

Uma importante contribuição de *GridWorld RL* é o aperfeiçoamento do controle da quantidade e do tempo entre diferentes iterações, disponível em diferentes botões e com valor que pode ser digitado pelo usuário. Soma-se a isso a adição de três modos de visualização, como apresenta a Figura 11 e a disponibilização de um gráfico sobre a convergência do modelo com o passar das iterações.

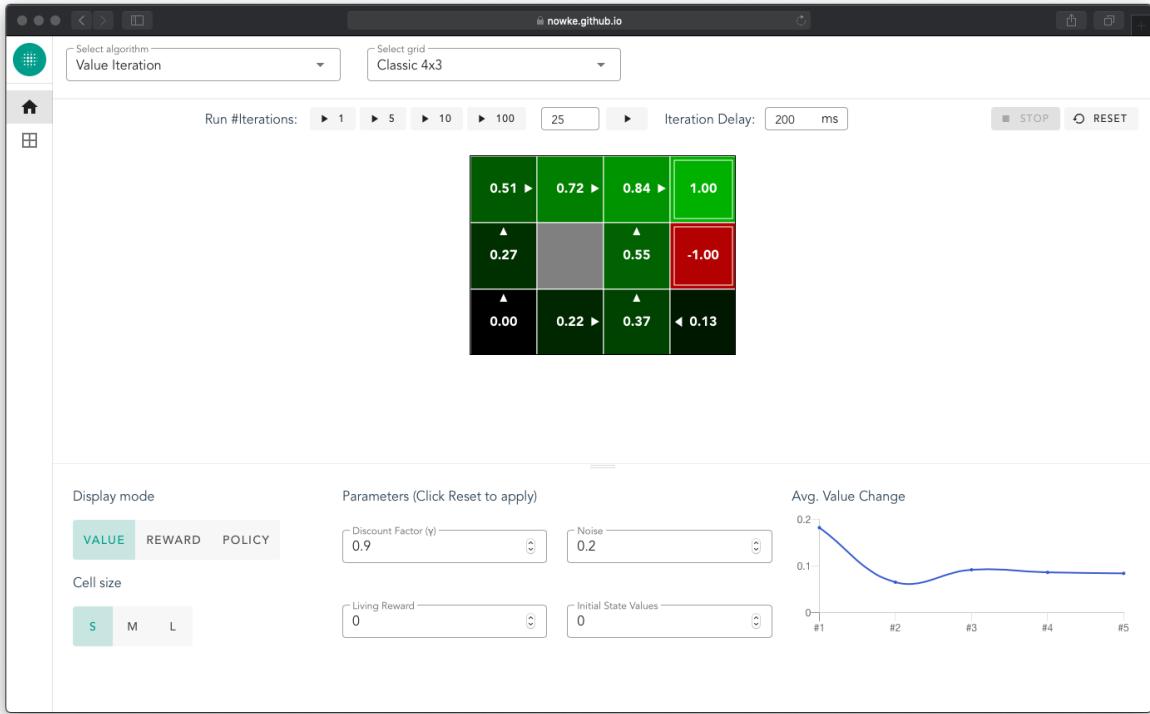


Figura 10: Interface principal da ferramenta *GridWorld RL*.

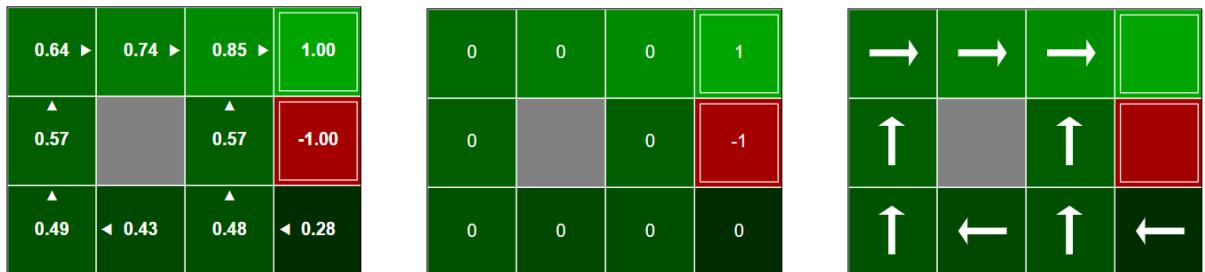


Figura 11: Modos de visualização na ferramenta *GridWorld RL*.

Apesar de todas suas contribuições, muitas delas inéditas, *GridWorld RL* se limita como ferramenta de aprendizado por dar suporte somente a algoritmos de Programação Dinâmica.

Outro trabalho também recente e com similar escopo foi desenvolvido por Xie *et al.* (2019). Neste projeto, denominado *QUBE*, o autor propõe ajudar *game designers* a preencherem lacunas referentes as suas formações técnicas através do uso de visualizações interativas na introdução de conceitos de Aprendizado de Máquina. A interface desta ferramenta é apresentada na Figura 12 e se assemelha aos dois trabalhos citados anteriormente nesta seção, tanto pelo aspecto visual quanto pelas funcionalidades.

Apesar de ser semelhante em muitos aspectos, o trabalho de Xie *et al.* (2019) se destaca no ponto didático ao disponibilizar um conteúdo educacional, na forma de tutoriais. Este trabalho, entretanto, não se encontra hoje disponível para acesso, nem possui código aberto.

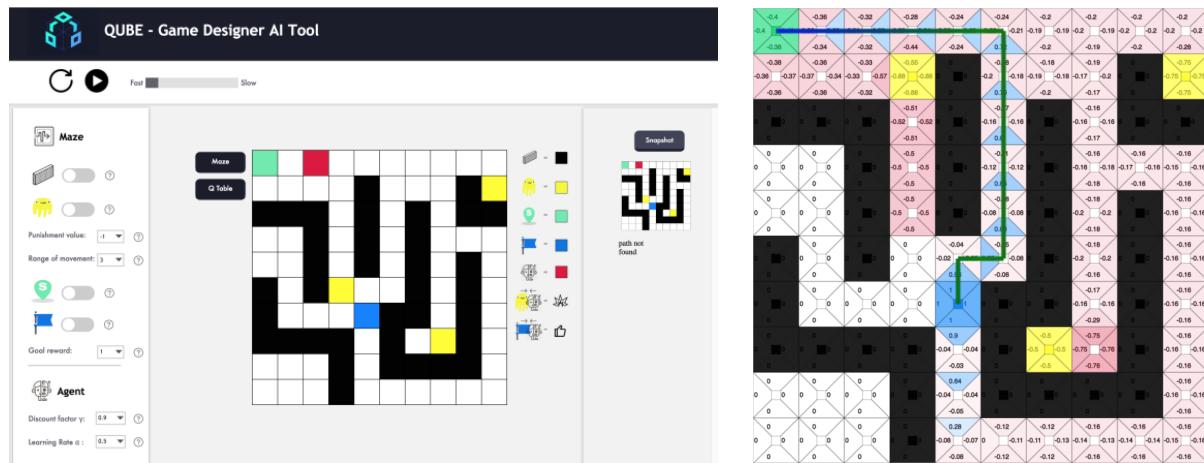


Figura 12: Interface principal da ferramenta QUBE (Xie et al., 2019).

4 Objetivos

Nesta seção apresentamos o objetivo geral e os objetivos específicos do projeto.

4.1 Objetivo geral

O objetivo geral deste projeto é fornecer recursos computacionais de apoio ao ensino de Aprendizado de Máquina, com ênfase nos conceitos básicos de Aprendizado por Reforço, que reúna funcionalidades essenciais dos trabalhos discutidos na Seção 3 em um único lugar e explore novas possibilidades de interação e ludicidade.

Além de recursos computacionais, pretende-se disponibilizar todo o código implementado e documentação realizada por meio deste relatório. Espera-se, desta forma, atingir um público que possua um conhecimento mínimo em computação, preferencialmente alunos que estejam cursando disciplinas de Inteligência Artificial ou Aprendizado de Máquina, e discentes à procura de material de apoio.

4.2 Objetivos específicos

Como recurso computacional, propomos neste projeto uma ferramenta gráfica, interativa e lúdica de visualização de métodos de Aprendizado por Reforço, de código aberto e disponível para uso na *web*. A ferramenta utiliza um ambiente tridimensional de simulação do *gridworld*, desenvolvido através do motor gráfico *Unity*. Dentre as funcionalidades, existentes em projetos anteriores ou inéditas, que este projeto deve suportar, estão:

- Demonstração do funcionamento dos algoritmos de Programação Dinâmica (Iteração de Política e Iteração de Valor) e Diferença Temporal (SARSA e Aprendizagem-Q);
- Personalização do *grid*, através da adição, remoção ou alteração das células em qualquer estado de execução do algoritmo. Para cada célula, espera-se modificar valores de recompensa, utilidade, políticas ou a característica de estado final ou inicial;
- Controle da quantidade de iterações, e do tempo entre as iterações na execução dos algoritmos, bem como suporte ao retrocesso das atualizações feitas;
- Modificação de parâmetros durante a execução do algoritmo;
- Disponibilidade de diferentes formas de visualização das células, com foco nas recompensas, valores ou políticas computadas, e um controle simples para redimensionamento do campo de visão.

Também é esperado, como resultado da conclusão deste projeto, que as implementações dos algoritmos desta ferramenta gráfica, feitas na linguagem de programação C#, possam ser utilizada para fins didáticos em conjunto com esta documentação.

5 Método

Existem atualmente ferramentas de diferentes naturezas disponíveis para o desenvolvimento de *software* interativo. Com a ascensão da internet, e a necessidade de se disponibilizar conteúdo de maneira simples e rápida através dela, se tornaram muito comuns abordagens mais voltadas para a *web*. Os projetos correlatos apresentados na Seção 3 são exemplos disso. Eles, entretanto, se apoiam em uma interface gráfica mais clássica, com componentes visuais bidimensionais. Neste projeto, por outro lado, buscamos implementar uma ferramenta de interface gráfica tridimensional e utilizar de recursos lúdicos mais interativos, trazendo, como resultado, uma experiência *gamificada*. Apesar de existirem *frameworks* e bibliotecas, muitas delas de código-aberto, disponíveis para o desenvolvimento *web*, tornou-se natural a adoção de uma plataforma com recursos adequados e dotada de um poder gráfico maior. Nesta seção, apresentamos estes recursos utilizados, bem como modelagens dos componentes e dos recursos visuais feitas.

5.1 Plataforma de desenvolvimento *Unity*

Para a construção da ferramenta de visualização proposta neste projeto, utilizamos a plataforma de desenvolvimento de jogos *Unity*, que traz consigo não somente um motor gráfico de jogos próprio, mas também um ambiente completo de desenvolvimento responsável por abstrair recursos avançados de renderização e controle de objetos (Haas, 2014). Tal plataforma também disponibiliza uma variedade de recursos extras através de uma loja online integrada, muitos deles gratuitos e mantidos inclusive pela comunidade, como texturas, animações, modelos, extensões de editor e até exemplos de projetos.

Apesar de a escolha de bons modelos e texturas ser uma parte essencial da execução deste projeto, por refletir na experiência final do usuário, esta etapa foi executada somente ao final. O uso de modelos geométricos pré-existentes, como cubos e retângulos, por sua vez, possibilitou no início um foco maior na arquitetura dos componentes principais responsáveis por controlar os elementos e a interface gráfica.

O controle dos elementos gráficos é feito por meio de *scripts*. As classes implementadas nestes *scripts* funcionam de maneira independente e possuem um ciclo de vida bem definido quando anexados a algum objeto. Para tanto, é necessário estender a classe-base *MonoBehaviour*, que define uma série de funções, acionadas em momentos distintos do programa, em uma arquitetura baseada a eventos.

É válido ressaltar também que esses *scripts* podem referenciar todo e qualquer elemento existente em tempo de execução. Isso pode ser feito com elementos definidos antes da inicialização do programa, ou de maneira dinâmica através da procura do elemento por nome ou tipo. Através das chamadas destas diferentes funções definidas pelo *Mono-*

Behavior, é possível modificar diferentes atributos destes elementos referenciados, como posicionamento no espaço, dimensionalidade ou aspectos mais complexos como cores, texturas e mesmo se o *script* de dado elemento deve ser desativado ou não.

5.2 Gerenciadores de estado

Mais do que simplesmente controlar objetos, podemos arquitetar os *scripts* para guardarem estados e também servirem como interface para entrada de dados pelo usuário através de elementos gráficos. Este foi um dos primeiros objetivos adotados na execução deste projeto.

Neste contexto, projetamos interfaces gráficas modulares responsáveis pelo controle de um conjunto de parâmetros correlatos, e introduzimos, para cada um desses módulos, uma classe gerenciadora. A Figura 13 apresenta interface gráfica já em seu estado final, composta por todos os módulos, e a relação de cada módulo com as classes gerenciadoras, representadas na cor cinza. São elas:

- **RlManager**, que computa os parâmetros utilizados diretamente nos algoritmos, como variáveis, tempo entre cada iteração e número de iterações, bem como controle de início, pausa e redefinição;
- **DeviationProbManager**, que computa e valida o movimento dos controles deslizantes referentes as probabilidades de movimento do agente;
- **DisplayModeManager**, referenciado nos scripts de cada estado, disponibilizando para eles o modo de exibição ativo;
- **InteractionModeManager**, responsável por alterar a interação do usuário com os objetos em cena para possibilitar criação, deleção e edição de estados. É responsável também por suprir dados do estado providos pelo usuário;
- **LocalizationLanguageManager**, que monitora o clique nos botões de cada idioma para atualizar os textos quando necessário.

Essas classes gerenciadoras traduzem as interações feitas no menu gráfico pelo usuário para os elementos existentes na cena, como os estados do *gridworld*, cada um controlado por seu *script* *GridBlock*, e a instância de *MDP* ativa, que referencia não somente os estados existentes, mas as dinâmicas do ambiente, como probabilidade do agente. Essa instância, por sua vez, é utilizada pelo algoritmo selecionado, em conjunto com uma classe que armazena todo o estado de execução, como número de iterações e se a execução foi finalizada ou não.

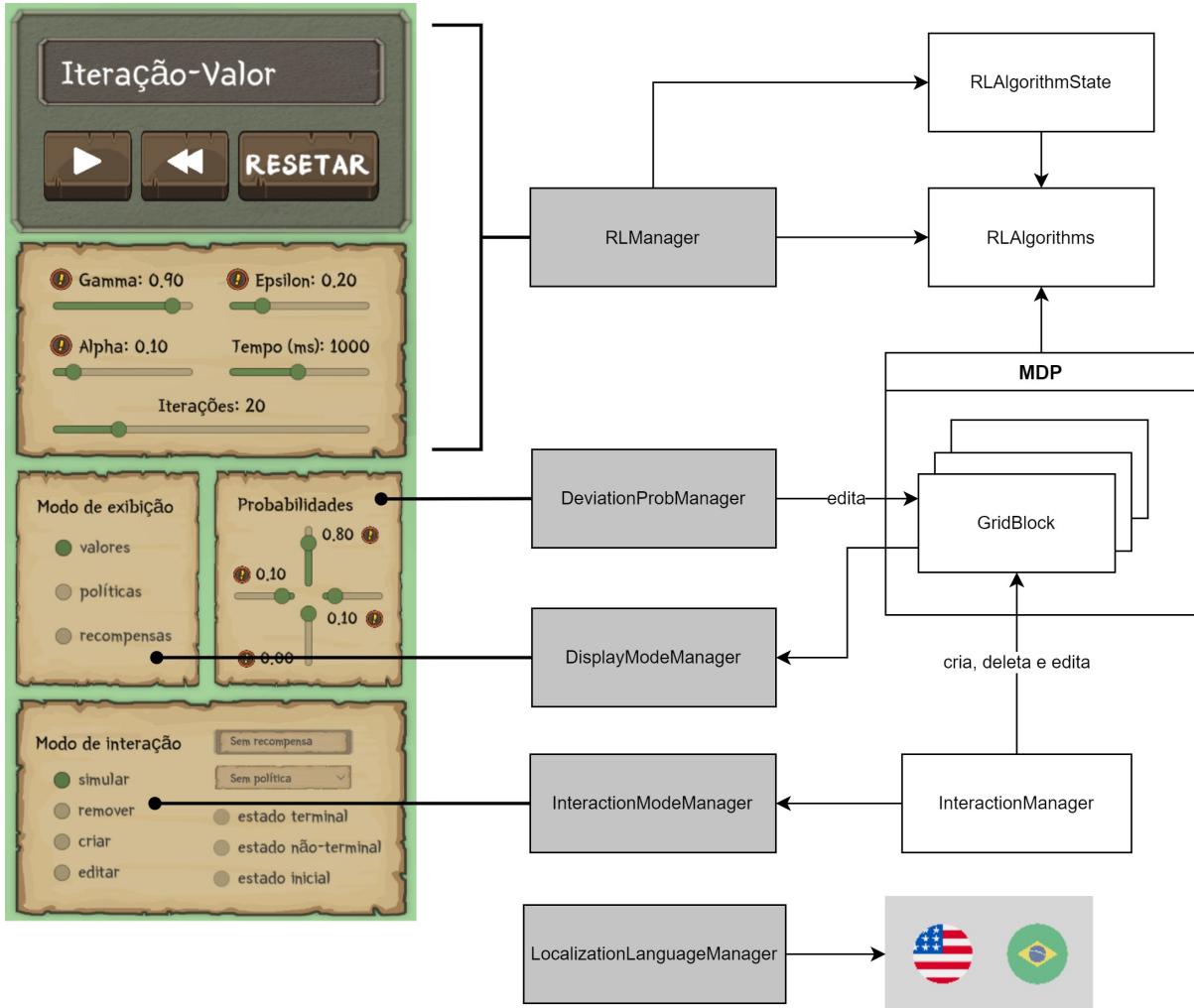


Figura 13: Módulos da interface gráfica e design dos principais componentes controladores.

A coleção completa de algoritmos está presente na classe *RLAlgorithms* através de métodos estáticos em forma de co-rotinas, que permitem a suspensão e resumo da execução a cada retorno, possibilitando iterações de duração e quantidade controladas. Co-rotinas, quando instanciadas, podem ser facilmente pausadas ou paradas. Isto reforça a necessidade de manter um objeto *RLAlgorithmState* para manter consistência entre diferentes execuções que podem ser paradas e ter o estado de execução local redefinido.

5.3 Modelagem do *MDP*

Para a representação do cenário e do *MDP*, utilizamos figuras geométricas simples, e texturas e modelos disponibilizados gratuitamente pela loja integrada do *Unity*.

Graficamente, cada estado do *MDP* é representado por blocos. Estes estados podem ser iniciais, terminais ou regulares, e são dotados de recompensas e de funções-valores, mesmo que ainda não computados. Todos esses elementos foram agregados em um modelo final reproduzível, denominado *Prefab*. A composição deste modelo e suas diferentes

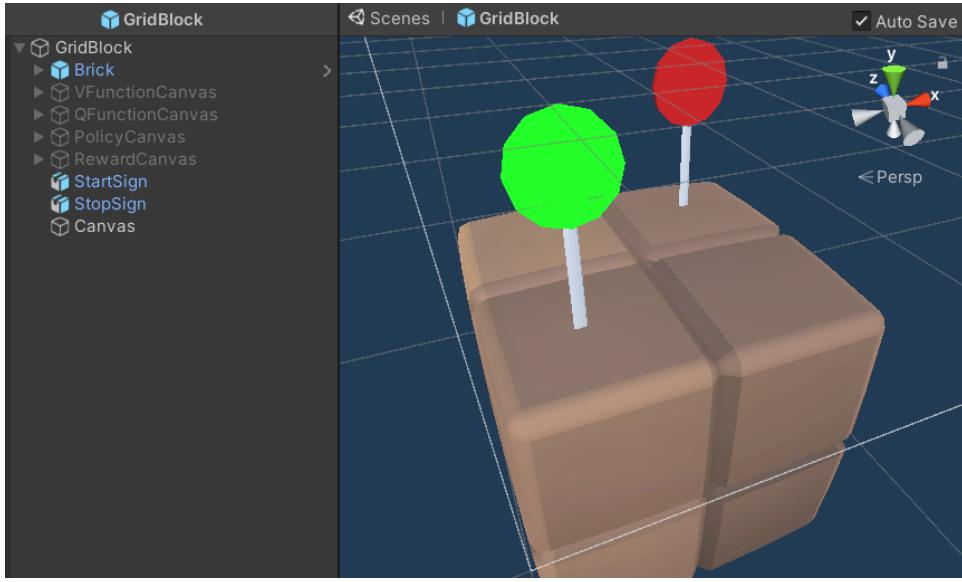


Figura 14: Modelagem do gridblock, que representa um estado do MDP.

configurações são mostradas na Figura 14. Essas configurações, bem como as informações computadas e mostradas, são controladas pelo *script GridBlock* apresentado na subseção anterior. Esta classe faz uma interface entre a instância do *MDP* existente e o elemento gráfico.

Utilizamos o padrão de projetos *singleton* para o gerenciador de *MDP*, obtendo, portanto, uma única instância dele a qualquer momento da execução. Como mostra a Figura 15, isso foi possível através da declaração de um método estático *getInstance* na classe *singleton*, que funciona como único construtor e deve ser, portanto, o único modo de obter a instância de sua própria classe, uma vez que o construtor, privado, é escondido do código cliente. Na primeira chamada deste método estático, a referência à estância é salva, e esse valor é reutilizado nas chamadas seguintes.

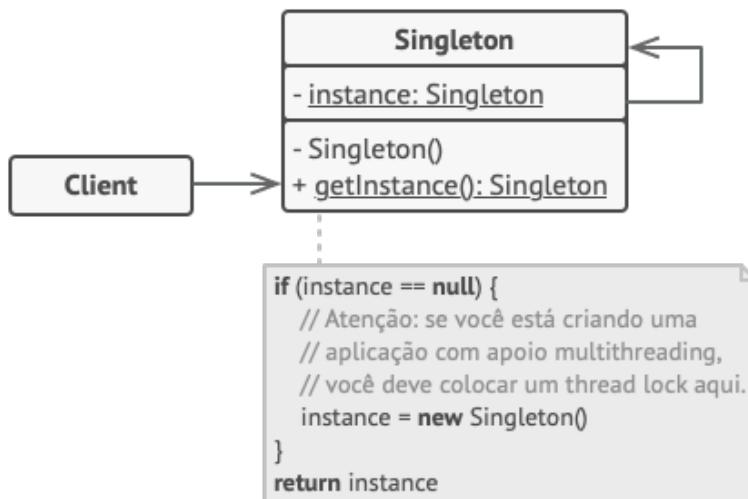


Figura 15: Estrutura de uma classe singleton ([Refactoring.Guru](#))

O *MDP* associado a esse gerenciador mantém referência das funções-valores (classes *QFunction* e *VFunction*) e política (classe *Policy*), e uma referência redundante dos estados a fim de otimização na validação de novos estados. Essas relações são apresentadas na Figura 16.

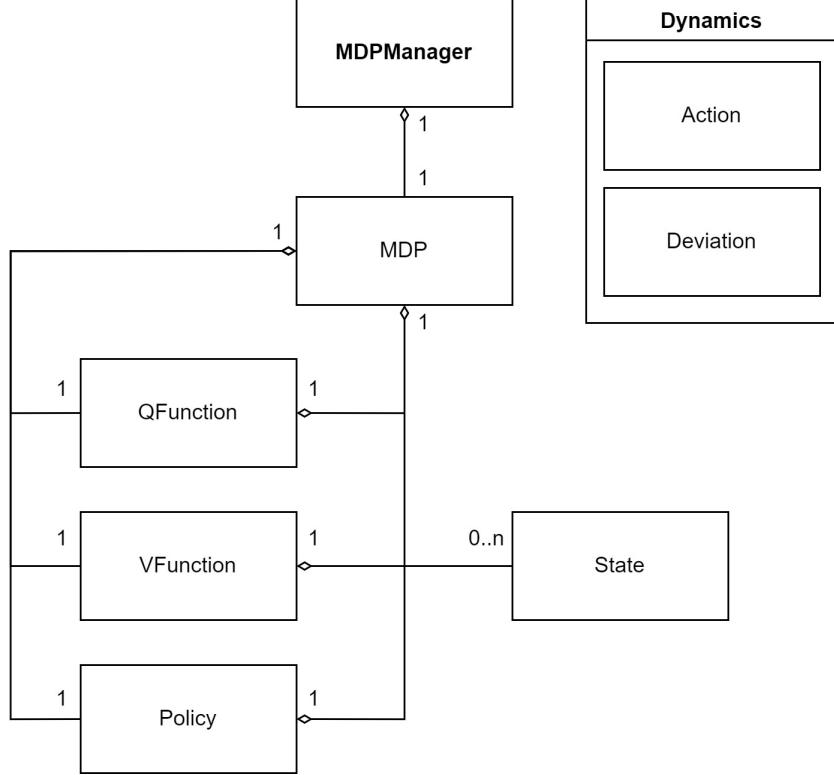


Figura 16: Modelagem dos principais componentes do *MDP*.

5.4 Distribuição do conteúdo

O *Unity* possibilita a disponibilização da aplicação desenvolvida em diferentes plataformas, com opções que vão além do comum *desktop*. De modo a distribuir a aplicação de maneira rápida e fácil para o maior público possível através da internet, aproveitamos o suporte do *Unity* ao *WebGL*.

WebGL, do inglês *Web Graphics Library*, é uma interface de programação do JavaScript utilizada para renderizar gráficos 3D e 2D dentro de um navegador *web* compatível sem o uso de plug-ins de terceiros (MDN contributors, 2021). Através dele, o *Unity* consegue gerar arquivos de uma aplicação *web* pronta para uso. A hospedagem desses arquivos foi feita através do serviço GitHub Pages e se encontra atualmente no endereço <https://igornerest.github.io/rviz-web/>. O código-fonte do projeto, por sua vez, pode acessado em <https://github.com/igornerest/RLViz>.

6 Resultados

Utilizando os conceitos e metodologias apresentados neste relatório, obtivemos a ferramenta interativa de visualização de agentes em ambiente de aprendizado por reforço apresentada na Figura 17. Até o momento, tratamos de uma visão mais geral das decisões tomadas e arquiteturas definidas. Vamos, desta forma, trazer nesta seção uma demonstração mais completa das funcionalidades implementadas, bem como as possibilidades de exploração e aprendizado que a ferramenta proporciona.



Figura 17: Ferramenta interativa de visualização de agentes em ambiente de aprendizado por reforço desenvolvida neste trabalho.

6.1 Funcionalidades principais da ferramenta

A ferramenta desenvolvida neste trabalho possui muitas das funcionalidades implementadas pelos trabalhos apresentados na Seção 3. Entre elas está a visualização do *gridworld* e dos diferentes atributos de cada estado, como política, recompensa e função-valor, e a personalização de diferentes parâmetros, que serão melhor explorados nas seguintes subseções. Soma-se a isso um conjunto de elementos inéditos incorporados.

Em sua inicialização, a ferramenta traz consigo a pré-configuração clássica do *gridworld* já discutida anteriormente na Seção 2.3. O pontapé inicial para a visualização se dá através de poucas interações. O primeiro módulo existente na interface gráfica apresenta uma caixa expansível com as opções de algoritmos tratados neste trabalho, em conjunto com as opções de inicializar, pausar ou mesmo regressar. Isso é mostrado na Figura 18.

Essa opção de regresso foi uma das novidades trazidas por este trabalho e consiste no uso de pilhas para armazenamento das políticas e funções-valores oriundas de cada interação.



Figura 18: Seleção de algoritmo e controle das execuções.

6.1.1 Parâmetros

A interface gráfica também é composta por módulos responsáveis pela personalização de diferentes parâmetros, que ditam as principais dinâmicas dos algoritmos, como mostra a Figura 19.



Figura 19: Personalização de parâmetros na ferramenta.

Entre os principais parâmetros disponíveis, estão:

- γ , fator de desconto que quantifica a importância dada para recompensas futuras;
- ϵ , probabilidade de realizar uma ação aleatória em vez de seguir a política atual (somente para Aprendizagem-Q e SARSA);
- α , taxa de aprendizado que representa a magnitude do passo que é dado em direção à solução, ou simplesmente o quanto se deve aceitar o novo computado;

- Número de iterações;
- Tempo entre interações;
- Probabilidades de o agente se mover para a direção desejada, desviar para os lados ou ir para trás.

6.1.2 Modos de interação

Mais dinâmicas do ambiente podem ser personalizadas através da modificação dos estados do *gridworld*. Para tanto, existem diferentes modos de interação específicos para criação, remoção e edição de estados, definidos através de uma caixa de seleção (Figura 20)

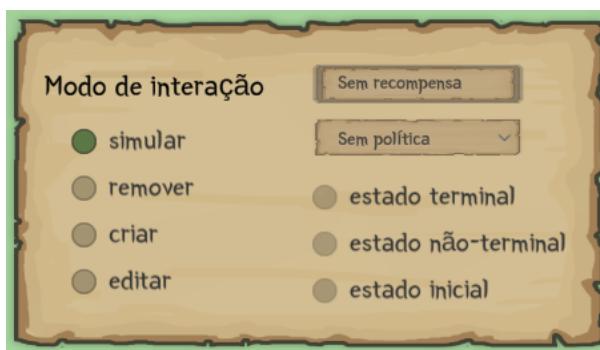


Figura 20: Modos de interação e configurações dos estados.

Os modos de remoção e edição funcionam de maneira similar em respeito a seleção de estados existentes em cena. Neste contexto, o bloco é selecionado pelo usuário através do cursor do *mouse*, e apresenta, em decorrência disto, uma textura transparente. A operação é confirmada com o clique do *mouse*. Já no modo de criação, um bloco transparente é mostrado em cena nas posições disponíveis, também através da seleção pelo cursor do *mouse*, bastando outro clique para conclusão da operação.

Durante a criação ou edição, o usuário pode modificar uma série de características e informações inerentes ao estado. Inclusive, funções-valores e políticas computadas através do aprendizado por reforço podem ser manualmente alteradas de maneira a explorar o comportamento do agente nestas circunstâncias. Esta ferramenta também permite a alteração da recompensa recebida pelo agente ao passar em determinado estado e a definição de estados como terminais ou iniciais, ditando, portanto, os episódios em algoritmos de Diferença Temporal. Todas essas configurações editáveis são mostradas também na Figura 20. Neste projeto, assumimos a existência de somente um estado inicial. Portanto, quando um novo estado inicial é instanciado, o existente se torna estado comum.

Por fim, de maneira a manter a consistência da execução dos algoritmos, adicionamos

um modo de simulação, que desabilita todos os outros campos de alteração. Da mesma forma, não é possível dar início a execução em outro modo.

6.1.3 Modos de visualização

De modo a proporcionar a melhor experiência de visualização de todos as informações inerentes aos estados, adicionamos três diferentes modos de visualização, mostrados na Figura 21. No primeiro modo, é possível observar as funções-valores e políticas computadas dividindo o mesmo espaço. No segundo modo, há um foco maior na política, enquanto no terceiro, somente nas recompensas.

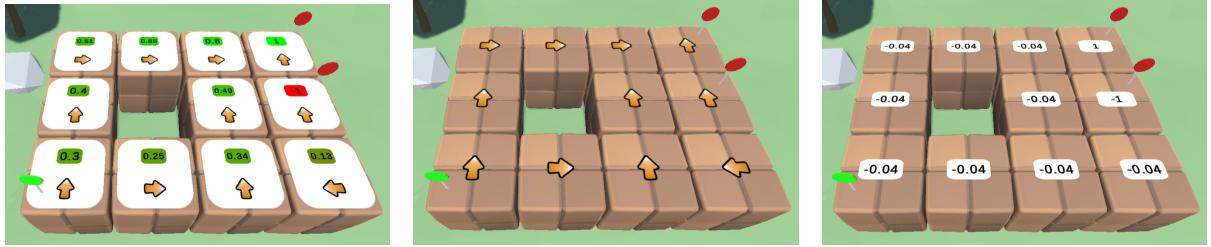


Figura 21: Modos de visualização na ferramenta.

6.2 Possibilidades de exploração

As funcionalidades principais apresentadas até o momento possibilitam diferentes experimentos. Nesta seção, demonstramos alguns exemplos.



Figura 22: Visualização do algoritmo de Iteração de Política a cada iteração.
A leitura das figuras deve ser feita da esquerda para a direita, de cima para baixo.

O principal experimento que pode ser realizado nesta ferramenta é a visualização dos diferentes algoritmos suportados. As Figuras 22 e 23, por exemplo, mostram a evolução do aprendizado para, respectivamente, os algoritmos de Iteração de Valor (a cada iteração) e Iteração de Política (a cada duas iterações). Podemos observar neste exemplo específico que a convergência do resultado ocorre de maneira mais rápida no primeiro algoritmo.

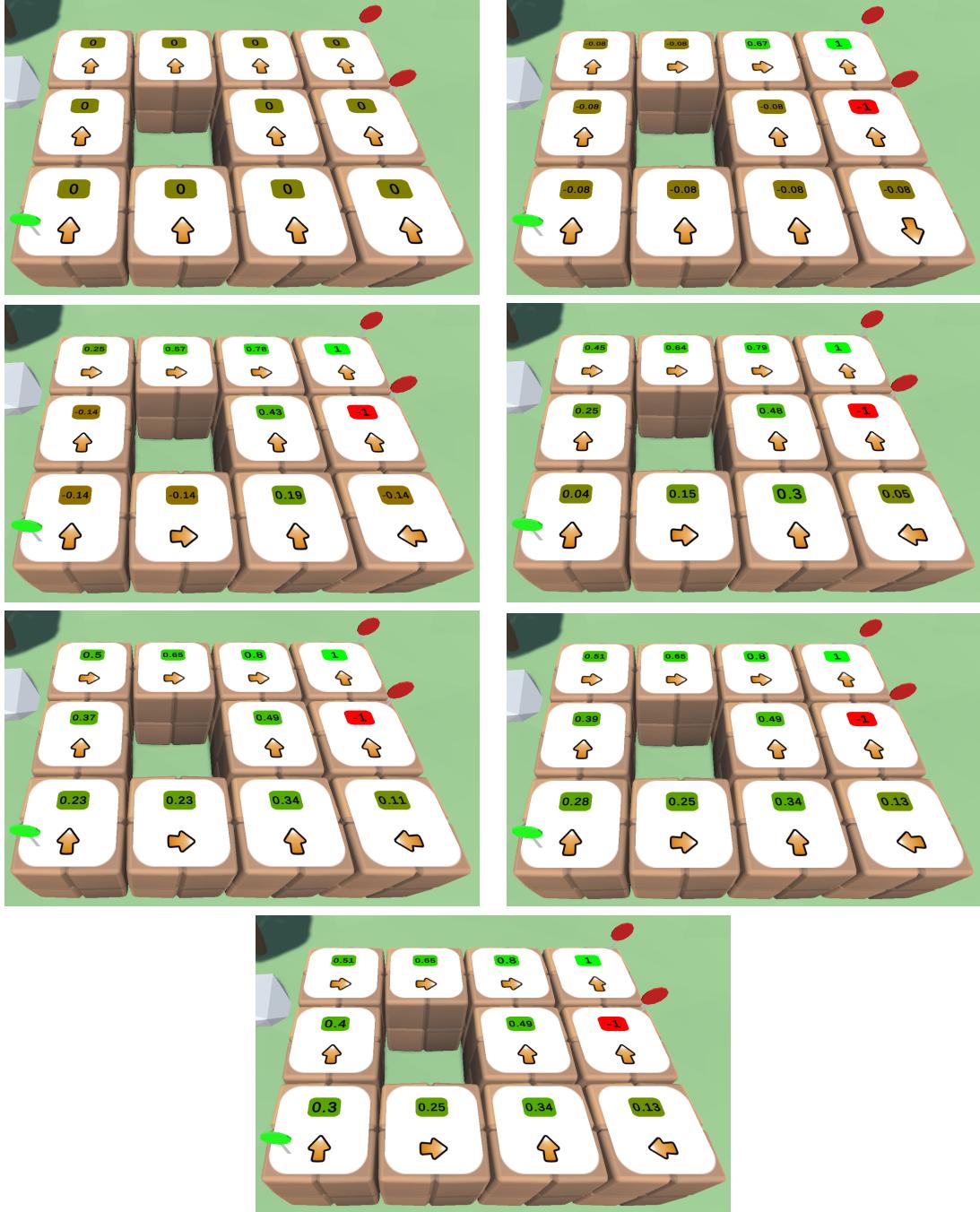


Figura 23: Visualização do algoritmo de Iteração de Valor a cada duas iterações. A leitura das figuras deve ser feita da esquerda para a direita, de cima para baixo.

As Figuras 24 e 25, por outro lado, demonstram algoritmos de Programação Dinâmica. Consideramos um passo maior, de 50 iterações, devido à convergência mais lenta.



Figura 24: Visualização do algoritmo de Aprendizagem- Q a cada 50 iterações.
A leitura das figuras deve ser feita da esquerda para a direita, de cima para baixo.



Figura 25: Visualização do algoritmo SARSA a cada 50 iterações.
A leitura das figuras deve ser feita da esquerda para a direita, de cima para baixo.

Nos experimentos para os algoritmos de Aprendizagem-Q e SARSA, o agente (em branco) se movimenta pelo *gridworld*, atualizando os estados por onde passa. Estes algoritmos, portanto, dependem da exploração e das decisões tomadas pelo agente, que, vale lembrar, possui um comportamento estocástico e dotado de incertezas. Isso ajuda a explicar o porquê de os resultados serem diferentes, mesmo após o mesmo número de iterações. Com um número maior de iterações, entretanto, pode-se esperar uma maior convergência e semelhança entre os resultados obtidos através das execuções dos dois algoritmos.

Além da visualização passo-a-passo relatada, podemos observar o reaprendizado dos agentes após mudanças feitas no ambiente. A Figura 26 apresenta um treinamento inicial no *gridworld*. Na Figura 27, removemos o estado na posição (3,2) e podemos observar que, ao executar novas iterações do algoritmo de aprendizado de máquina, o agente reaprende as políticas de maneira a chegar em seu objetivo final.

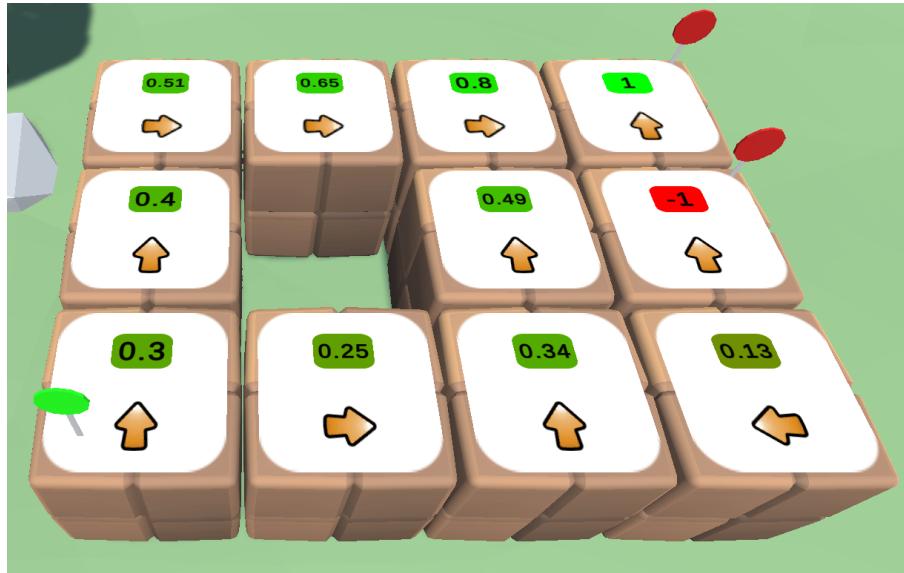


Figura 26: Aprendizado inicial do agente no *gridworld*.

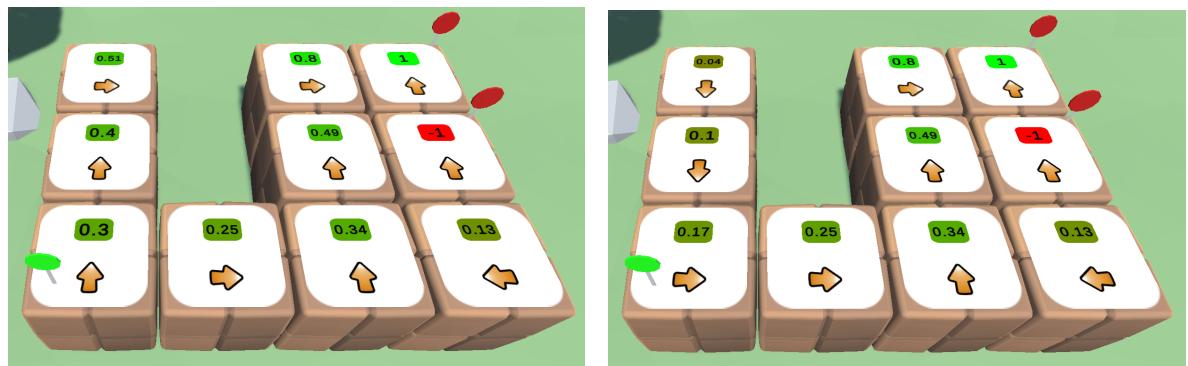


Figura 27: Adequação das políticas após remoção do estado na posição (3,2). Políticas são reordenadas a fim de evitar o caminho bloqueado

Na Figura 28, removemos o estado na posição (1,3) e adicionamos um na posição (2,2). Ao executar novas iterações do algoritmo de aprendizado de máquina, é possível novamente perceber que o agente ajusta as políticas de maneira a encontrar um caminho do estado inicial ao final.

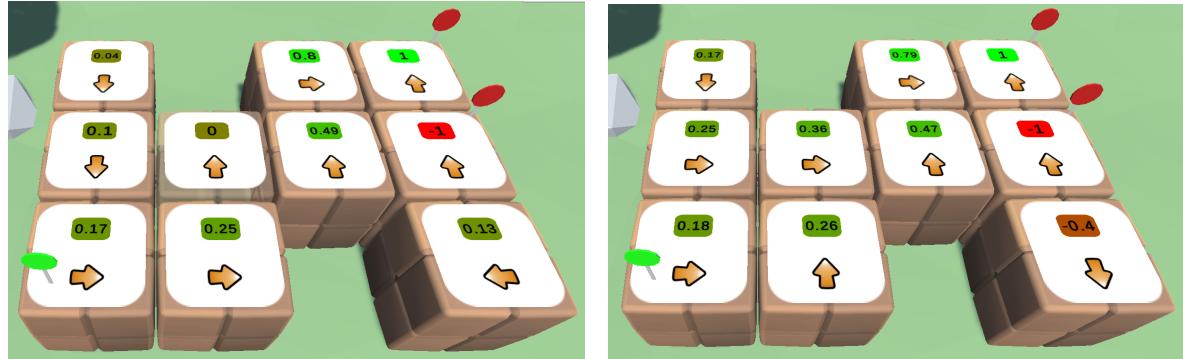


Figura 28: Adequação das políticas após remoção do estado na posição (1,3) e adição de outro na posição (2,2). Políticas são reordenadas a fim de evitar o caminho bloqueado

7 Conclusão

Com o rápido avanço da ciência e a grande aposta contemporânea em subáreas da Inteligência Artificial, se tornou muito comum atualmente a aplicação de técnicas cada vez mais sofisticadas para a resolução de problemas. Isso ocorre especialmente para técnicas de Aprendizado por Reforço. É natural que o uso destas técnicas, que se baseiam em um arcabouço teórico mais pesado, sejam acompanhadas de bibliotecas ou *frameworks* responsáveis por entregar uma maior abstração para desenvolvedores. É importante, entretanto, entender as raízes ou conceitos essenciais por trás destes algoritmos mais complexos. Neste contexto, este trabalho permitiu a criação de uma ferramenta de visualização de agentes em ambiente de Aprendizado por Reforço com alguns algoritmos básicos apresentados na literatura, de forma a contribuir para o aprendizado destes conceitos.

A ferramenta implementada neste trabalho se baseia na experiência já existente em trabalhos anteriormente citados, ao mesmo tempo que explora as deficiências e limitações deles através da adição de novas funcionalidades e possibilidades de exploração. O foco, em todo momento, foi dado à interatividade.

7.1 Contribuições

Umas das principais contribuições deste projeto é a agregação de uma variedade de funcionalidades e personalizações existentes em uma única ferramenta. O desenvolvimento deste projeto possibilitou o uso de algoritmos de diferentes naturezas, como Programação Dinâmica e Diferença Temporal, e a personalização de parâmetros responsáveis pelas dinâmicas tanto destes algoritmos, quanto do ambiente, através das probabilidades inerentes ao movimento do agente e das características de cada estado.

Outra contribuição relacionada ao trabalho é a deleção, criação e edição dos estados entre as diferentes execuções, possibilitando a análise do comportamento e adequação do agente frente a mudanças. Durante as execuções, é possível alterar parâmetros dos algoritmos. Soma-se a isso, a funcionalidade inédita de reverter as iterações de cada algoritmo, que permite o usuário a observação do agente em diferentes cenários.

É importante ainda citar que o trabalho, dada sua característica didática, foi desenvolvido tendo o fácil acesso como prioridade. Através do suporte à *WebGL*, foi possível a disponibilização de uma aplicação web. Para maior alcance de usuários à ferramenta, desenvolvemos um sistema simples de localização para o português e o inglês. Espera-se também que os conceitos descritos através deste trabalho também sirvam como referência didática.

7.2 Limitações e trabalhos futuros

A execução deste projeto contou com uma adequação do escopo, devido às limitações de tempo, e por se tratar de uma ferramenta feita do princípio. Portanto, apesar de todas as contribuições e novidades apresentadas, algumas funcionalidades que já existiam nos trabalhos correlatos não foram implementadas nesse primeiro momento.

Entre essas funcionalidades estão a informação gráfica da variação da função-valor em relação ao tempo e a possibilidade de salvar ou carregar mapas entre diferentes sessões, existentes no trabalho de Kumar (2020).

Outra contribuição futura é a adição de novos algoritmos, e possivelmente a expansão para conceitos de aprendizado profundo. Todas essas sugestões poderiam ser implementadas de maneira simples, visto que a arquitetura principal para controle de objetos em cena e a orquestração dos algoritmos já foi devidamente projetada.

Referências

- Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, *et al.* 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arxiv preprint arxiv:1603.04467*.
- Doya, Kenji. 1995. Temporal difference learning in continuous time and space. *Advances in neural information processing systems*, **8**, 1073–1079.
- Gevorkyan, Migran N, Demidova, Anastasia V, Demidova, Tatiana S, & Sobolev, Anton A. 2019. Review and comparative analysis of machine learning libraries for machine learning. *Discrete and continuous models and applied computational science*, **27**(4), 305–315.
- Haas, John K. 2014. A history of the unity game engine. *Diss. worcester polytechnic institute*, **483**, 484.
- Juliani, Arthur, Berges, Vincent-Pierre, Teng, Ervin, Cohen, Andrew, Harper, Jonathan, Elion, Chris, Goy, Chris, Gao, Yuan, Henry, Hunter, Mattar, Marwan, *et al.* 2018. Unity: A general platform for intelligent agents. *arxiv preprint arxiv:1809.02627*.
- Karpathy, Andrej. 2015. *REINFORCEjs*. <https://github.com/karpathy/reinforcejs>. Acessado pela última vez em 03 de Maio de 2022.
- Kumar, Navaneesh. 2020. *GridWorld RL*. <https://github.com/nowke/rlviz>. Acessado pela última vez em 03 de Maio de 2022.
- MDN contributors. 2021. *WebGL*. https://developer.mozilla.org/pt-BR/docs/Web/API/WebGL_API. Acessado pela última vez em 03 de Maio de 2022.
- Mohammed, Mohssen, Khan, Muhammad Badruddin, & Bashier, Eihab Bashier Mohammed. 2016. *Machine learning: algorithms and applications*. CRC Press.
- Refactoring.Guru. 2021. *Singleton*. <https://refactoring.guru/pt-br/design-patterns/singleton>. Acessado pela última vez em 27 de Junho de 2022.
- Russell, Stuart, & Norvig, Peter. 2002. *Artificial intelligence: A modern approach*. Pearson.
- Sutton, Richard S, & Barto, Andrew G. 2018. *Reinforcement learning: An introduction*. MIT press.

Xie, Jiachi, Myers, Chelsea M, & Zhu, Jichen. 2019. Interactive visualizer to facilitate game designers in understanding machine learning. *Pages 1–6 of: Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems.*

Yarlagadda, Ravi Teja. 2018. Internet of things & artificial intelligence in modern society. *International Journal of Creative Research Thoughts (IJCRT), ISSN*, 2320–2882.