Frozen Lake via Q-Learning modificado
Exame de CT213 - Inteligência Artificial para Robótica Móvel

Lucca Moura Zoppi Maia[1] e Nando Ferreira Farias[2]

I. Introdução Teórica

No contexto do aprendizado por reforço livre de modelo, o algoritmo *Q-Learning* permite o aprendizado de uma política ótima π determinística diretamente através da estimativa da função ação-valor de um Processo Decisório de Markov (MDP). O aprendizado é *off-policy*, com exploração realizada através de uma política ε*-greedy i.e.* uma política quase determinística que escolhe uma ação aleatória com probabilidade ε ou uma ação *greedy*, que consiste da ação a qual se atribui maior valor para dado estado.

Para isso, é necessário que o espaço de estados e o de ações sejam discretizados, para que cada um corresponda a um conjunto finito e a função ação valor possa ser representada através de uma tabela, doravante chamada tabela Q. Esse aspecto representa uma desvantagem caso o espaço de estados ou de ações tenha tamanho considerável, pois a tabela Q então exigiria uma quantidade inviável de memória.

Para suplementar essa desvantagem, foi concebido o algoritmo *Deep Q-Learning* (DQN), que utiliza de uma rede neural profunda para aproximação da tabela Q, e aplica *experience replay* e *fixed Q-targets* para estabilização do treinamento dessa rede.

Experience replay consiste de armazenar tuplas $(S_t, A_t, R_{t+1}, S_{t+1})$ (estado, ação, recompensa, próximo estado no tempo t) em um *buffer* do qual são amostrados *mini-batches* aleatórios utilizados no treinamento da rede. Enquanto *fixed Q-targets* se trata de utilizar os pesos da rede fixos em seus valores anteriores a cada iteração de treinamento para gerar os *targets*, que representam estimativas atualizadas da tabela Q, usados no treinamento da rede. Com isso, espera-se melhorar a convergência da rede.

Esse aspecto de estabilização do *experience replay* pode ser aproveitado no algoritmo *Q-Learning* original, criando-se um *replay buffer* utilizado para amostrar os *mini-batches* a partir dos quais os valores da tabela Q são atualizados.

O erro de representação da rede neural presente no DQN é substituído pelo hiperparâmetro α do *Q-Learning*, que cumpre o papel de taxa de aprendizado. A taxa de aprendizado então é utilizada em uma média móvel na atualização da tabela Q, segundo (1).

$$\begin{cases} Q(S_t, A_t) = Q(S_t, A_t) + \alpha \cdot \delta_t \\ \delta_t = R_t + \gamma \cdot \max\limits_{a' \in A} \{Q(S_{t+1}, a')\} - Q(S_t, A_t) \end{cases} \quad (1)$$

[1]lucca.maia@ga.ita.br
[2]nando.farias@ga.ita.br

em que A é o espaço de ações discretizado, $(S_t, A_t, R_{t+1}, S_{t+1})$ é uma experiência registrada no passo de tempo t, δ_t é o $target$ gerado para aquela experiência individual e γ é o fator de desconto.

Além disso, pode-se também adaptar a ideia dos *fixed Q-targets*, atualizando-se a tabela Q com *targets* gerados a partir de cada *mini-batch* de experiência e dos valores da tabela anteriores à atualização em andamento.

A. Problema a ser resolvido

O problema explorado no presente trabalho é o *Frozen Lake* disponível entre os ambientes *Toy Text* do *framework Gymnasyum* de `Python`.

Esse problema consiste de um MDP no qual um agente deve aprender uma política que o leve de sua posição inicial, no canto superior esquerdo do tabuleiro, até seu objetivo, no canto inferior direito, como ilustrado em 1.

Fig. 1: Exemplo de tabuleiro 4 x 4 do *Frozen Lake*.

O tabuleiro é 4-conectado e seus limites são paredes, de modo que, em cada posição, o agente pode decidir entre as ações de 0 a 3, correspondentes, em ordem crescente, a ir para esquerda, para baixo, para direita e para cima, e o movimento ocorre se a direção não for contrária a uma das paredes. Cada posição corresponde a uma possível observação (estado) que são representadas pelos índices da matriz de posições achatada.

Ademais, ao longo do tabuleiro há buracos no gelo, de modo que o agente cai quando passa sobre um deles, interrompendo o episódio. Além disso, o ambiente permite escolher entre tornar o chão escorregadio ou não. No último caso, sob qualquer política determinística, o MDP se converte em um Processo de Markov com Recompensa (MRP) determinístico. Por outro lado, se o chão é escorregadio,

então o agente tem apenas $33,\bar{3}\%$ de chance de ir na direção escolhida, e, de outro modo, probabilidades iguais de ir em uma das direções perpendiculares.

A figura 2 ilustra o Processo de Markov descrito, exemplificando a transição de um estado n, para seus estados adjacentes, com as probabilidades de transição anotadas, assumindo que tenha sido escolhido ir para baixo e que o estado n não seja adjacente a uma parede.

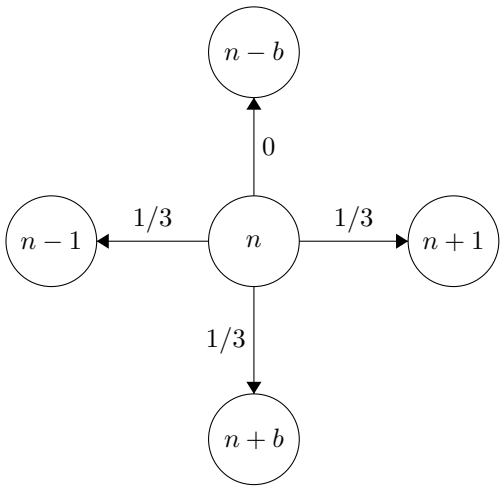

Fig. 2: Probabilidades de transição a partir de um estado n quando se escolhe ir para baixo, em que b é o lado do tabuleiro.

Finalmente, as recompensas geradas pelo ambiente são 0 enquanto o agente não atinge o objetivo, incluindo caso caia em um buraco, e 1 ao atingir o objetivo.

II. Implementação

A implementação desenvolvida assim como instruções de uso se encontram disponíveis no repositório público em Exame-CT213.

O agente é representado pela classe `DQNAgent`, que guarda os hiperparâmetros α e γ do *Q-Learning*, assim como o ε da política exploratória, sua taxa de decaimento, seu valor mínimo e o *replay buffer* na forma de um `deque` com tamanho máximo especificado, tal que descarta seus valores mais antigos quando atinge esse máximo. A tabela Q é um `NumPy Array` de formato (#observações, #ações), *i.e.* número de posições no tabuleiro por número de ações possíveis, inicializado com zeros.

Esse agente possui o método `act` para geração de ações ε*-greedy* a partir de um estado e de seus valores internos de ε e da tabela Q.

Além disso, possui `append_experience` para adicionar experiências individuais ao *replay buffer* e `replay` para aprender a partir de um número especificado `batch_size` de experiências passadas, aplicando a técnica de *fixed Q-targets*, que mantém a tabela Q fixa antes de atualizar seus valores com base nos `batch_size` *targets* gerados.

Finalmente, foi implementado um método `update` para permitir o aprendizado a partir da experiência individual mais recente. Isso foi feito por motivos discutidos na seção §III.

As recompensas fornecidas ao agente passam por modificações em relação às fornecidas pelo ambiente, segundo detalhado em §III.

A operação do código se baseia em definir dois parâmetros do ambiente `BOARD_SZ` e `SLIPPERY`, que definem, respectivamente, o lado do tabuleiro e se o chão é escorregadio ou não. Então, deve ser executado o *script* `train_agent.py` para o treinamento do agente sobre um tabuleiro gerado aleatoriamente a partir de uma *seed* fixa.

Esse *script* escreve no terminal informações como número de episódios transcorridos, duração em passos de tempo de cada episódio e o retorno acumulado por episódio, ao longo de sua execução. Ao terminar, exibe a política *greedy* obtida, representando as direções escolhidas pelas suas iniciais L(eft), D(own), R(ight), U(p) e deixando vazias posições nas quais os valores da tabela Q não foram atualizados durante o treinamento *i.e.* permaneceram em 0.

Após o treinamento, usa-se `evaluate.py` para se obter resultados acerca do agente com a *greedy policy* obtida, considerando-se um ambiente com parâmetros idênticos aos do treinamento.

III. Resultados e Discussões

Inicialmente, considerou-se o problema mais fácil, em que o chão não é escorregadio, e realizou-se o *reward engineering* de modo a obter-se recompensas intermediárias que contribuíssem para o agente encontrar uma política que o levasse ao objetivo. Com base em testes com tabuleiros gerados por diferentes *seeds* e com diferentes valores de lado, concebeu-se o sistema de recompensas e punições representado na Tabela I, que descreve as situações em que as recompensas ou punições são aplicadas e seus pesos - positivos para recompensas e negativos para punições.

TABLE I: Recompensas e punições dadas ao agente durante o treino.

Condição	Peso
Atingiu Objetivo	1000
Caiu em buraco	-700
Escolheu direção de buraco	-700
Escolheu $\downarrow$ ou $\rightarrow$	2
Executou $\downarrow$ ou $\rightarrow$	0.5
Escolheu $\uparrow$ ou $\leftarrow$	-5
Ficou parado	-2.5
Escolheu retroceder	-1
Ação deliberada	1

Além disso, foi imposto um limite de tempo por episódio, proporcional ao lado do tabuleiro, de modo que, para o tabuleiro 4 x 4, o limite é 100 passos de tempo, enquanto para o 8 x 8 esse limite é 200. Com isso o agente treinado por 2000 episódios conseguiu, para uma *seed* fixa em particular (0) resolver o problema sem chão escorregadio com tabuleiro de tamanho até pelo menos 30 x 30 - não foram feitos testes além disso pela execução demorada. Sendo determinístico

quando o chão é não escorregadio, não há necessidade de considerar múltiplas avaliações, pois a taxa de sucesso é sempre 100 %.

Nota-se que algumas das condições da Tabela I foram inseridas pensando-se no problema com chão escorregadio, no qual há diferença entre "ir em direção a" e "escolher ir em direção a".

Então, considerou-se o problema com chão escorregadio. Novamente, foi fixada a *seed* 0, e foram feitas avaliações ao longo de 1000 episódios do agente treinado por 2000 episódios. A necessidade de se fazer a avaliação por múltiplos episódios se deve ao fato de que, para uma dada política determinística, o agente pode ter probabilidade diferente de 0 e de 100 % de ter sucesso. Além disso, foi adicionada mais uma recompensa, que atrapalharia o desempenho do agente treinado sobre mapas com chão não escorregadio: o agente recebe uma recompensa com peso 10 por atingir o tempo limite. A razão disso é que deseja-se valorizar políticas que levem o agente a evitar buracos por mais tempo, o que, enquanto no problema de chão não escorregadio poderia levá-lo a ficar parado ou em *loop*, no problema com chão escorregadio, há situações em que existem abordagens seguras mas que levam o agente a perder no tempo parte das vezes.

Com isso para uma execução em particular de treino + avaliação para cada um dos tamanhos de tabuleiro, foram obtidas as taxas de sucesso representadas na Tabela II.

TABLE II: Taxas de sucesso ao longo de 1000 episódios por tamanho de tabuleiro.

Lado	Taxa de sucesso	Taxa de perda por tempo limite
4	99.20%	0.80%
6	71.70%	10.80%
8	2.40%	0.30%

Na Tabela II, taxa de perda por tempo significa ter o episódio terminado por atingir o tempo limite, em vez de por cair em um buraco. Portanto, nota-se que a política aprendida para o tabuleiro 4 x 4 é totalmente segura em termos de evitar buracos, e todos os episódios não completos se devem a uma pequena chance de não chegar ao objetivo a tempo. Já no tabuleiro 6x6, vê-se que em 17.5% dos episódios, o agente caiu em um buraco, enquanto a taxa de sucesso diminui consideravelmente em relação ao tabuleiro menor. Finalmente, com o tabuleiro 8x8, se torna muito raro que o agente chegue ao seu objetivo, e em quase todas as vezes em que falha, esse caiu em algum buraco.

Observa-se que, a depender da execução do treino, há certa variação da política aprendida, o que se deve à grande aleatoriedade envolvida no problema com chão escorregadio. Por outro lado, seria relativamente fácil obter sistemas de recompensas e punições otimizados para mapas específicos de tamanhos específicos, pois cada mapa tem suas particularidades *e.g.* há mapas com caminhos que permitem abordagens totalmente seguras, que evitam completamente cair em buracos, como no caso do tabuleiro 4 x 4 considerado, mas há outros em isso não é possível, pois, por exemplo, há posições no tabuleiro adjacentes a 2 buracos. No entanto, optou-se por obter-se um sistema que apresenta-se um desempenho razoável para mapas diversos.

Além das taxas de sucesso, podem-se observar os históricos de retorno descontado ($\gamma = 0.99$) para o treino em cada uma dessas execuções. Nota-se que nessas execuções foi utilizado um ε inicial de 0.8, taxa de decaimento de 0.99 e ε mínimo de 0.01.

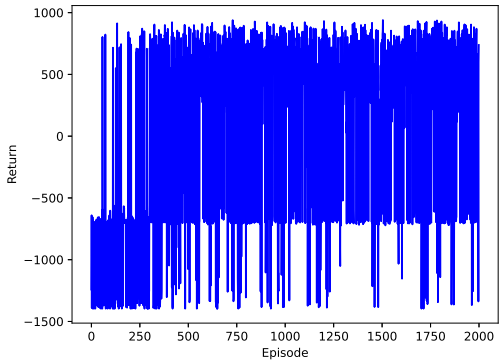

Fig. 3: Histórico de retorno no treino para o tabuleiro 4x4.

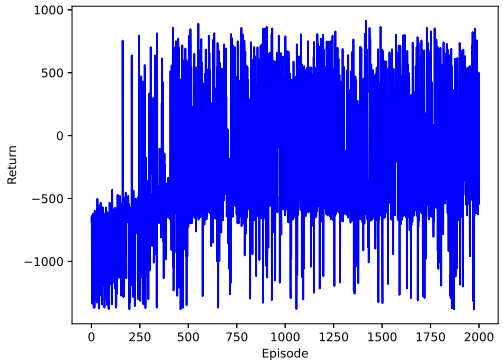

Fig. 4: Histórico de retorno no treino para o tabuleiro 6x6.

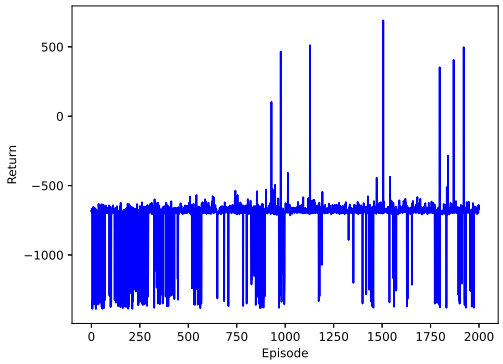

Fig. 5: Histórico de retorno no treino para o tabuleiro 8x8.

Nas figuras 3, 4 e 5, pode-se verificar o claro crescimento da dificuldade de se obter uma política com retorno razoável.

Além disso, podem ser observadas as políticas *greedy* aprendidas em cada caso.

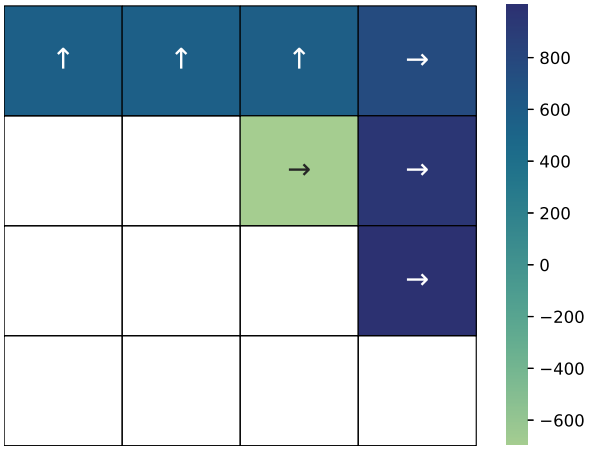

Fig. 6: Política *greedy* representada em um *heatmap* com as setas indicando as direções de preferência.

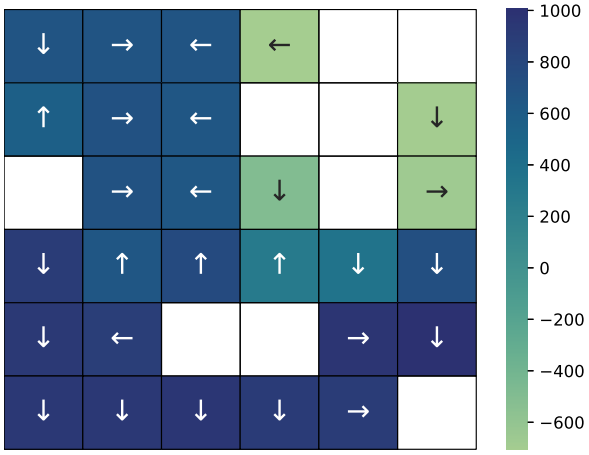

Fig. 7: Política *greedy* aprendida com tabuleiro 6 x 6.

Nos mapas de calor das figuras 6, 7 e 8, as posições em branco correspondem a estados não inicializados *i.e.* durante o treino, o agente ou nunca esteve na posição ou o episódio terminou ao atingi-la. Portanto, todos os buracos estão em branco, assim como o objetivo, que, como dito anteriormente, sempre se encontra no canto inferior direito.

Assim, vê-se nas políticas aprendidas a clara tendência de se evitar ir em direção aos buracos. Em particular, a política aprendida pelo agente no tabuleiro 4 x 4 corresponde à abordagem segura para esse mapa e difere consideravelmente da política aprendida com o chão não escorregadio, que pode ser observada em 9.

Nota-se também que a posição (1, 2) (coordenadas como índices em uma matriz começando em 0) difere drasticamente de valor entre os dois ambientes, o que se deve à não existência de direção segura nessa posição com o chão escorregadio, pois é adjacente a dois buracos, como mostra

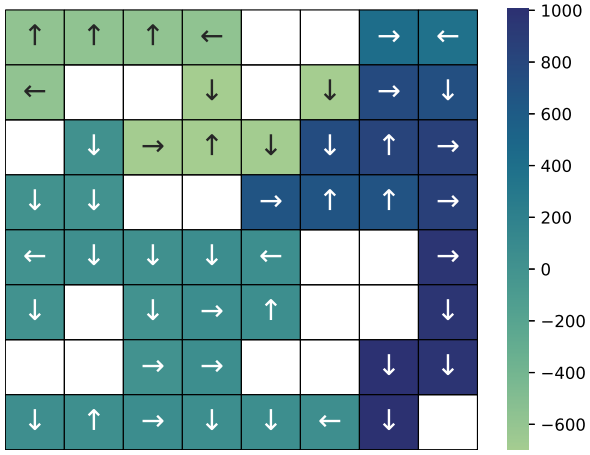

Fig. 8: Política *greedy* aprendida com tabuleiro 8 x 8.

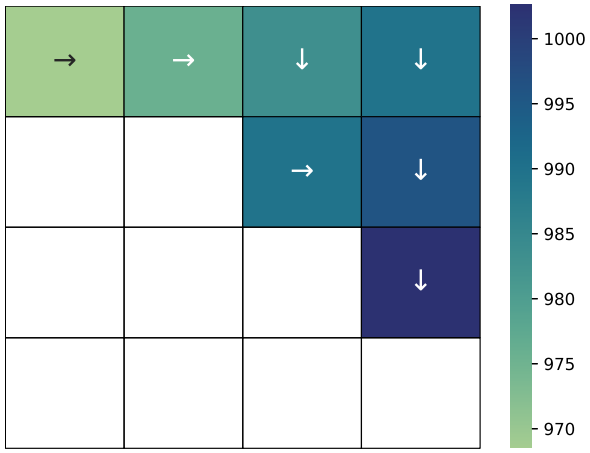

Fig. 9: Política *greedy* aprendida no mesmo mapa, mas com chão não escorregadio.

o *frame* final de avaliação do agente nos testes com chão escorregadio.

Por outro lado, para o tabuleiro 6 x 6, considera-se o mapa utilizado em 11.

Vê-se que a política aprendida em 7 evita quase totalmente direções que podem levar o agente a cair em um buraco, exceto na posição (2,3) onde há adjacência a 2 buracos. Desse modo, não fosse o limite de tempo, com base na Tabela II, a taxa de sucesso poderia se tornar 80%+, com a chance de cair em um buraco consideravelmente reduzida. Já a política aprendida com chão não escorregadio difere consideravelmente, já que não há necessidade de evitar direções inseguras.

Finalmente, para o tabuleiro 8 x 8, consideram-se o mapa, o histórico de retorno durante o treino e a política aprendida com chão não escorregadio.

Na comparação entre os históricos em 5 e 14, vê-se que a política ε-*greedy* ao final produz um valor de retorno consistentemente alto no último caso, o que é inverso para o caso do chão escorregadio. Além disso, do mapa em 13,

Fig. 10: Foto do tabuleiro 4 x 4 considerado.

Fig. 11: Foto do tabuleiro 6 x 6 considerado.

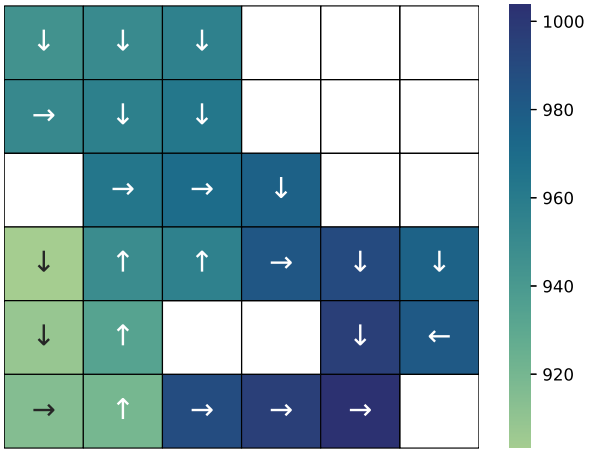

Fig. 12: Política *greedy* aprendida no tabuleiro 6 x 6 com chão não escorregadio.

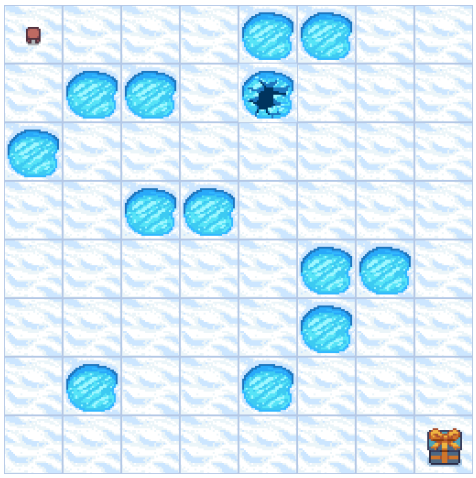

Fig. 13: Foto do tabuleiro 8 x 8 considerado, durante avaliação com chão escorregadio.

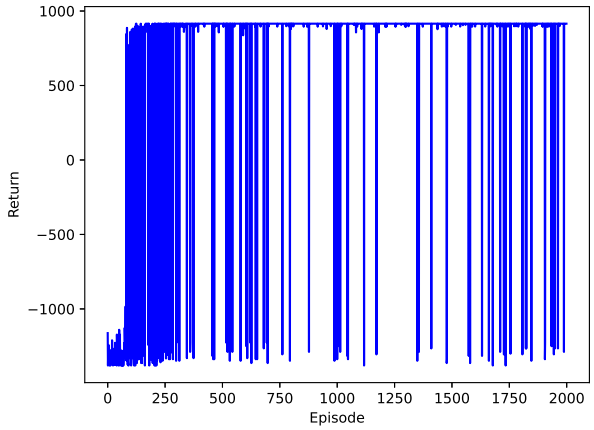

Fig. 14: Histórico de retorno no treino para o tabuleiro 8x8 com chão não escorregadio.

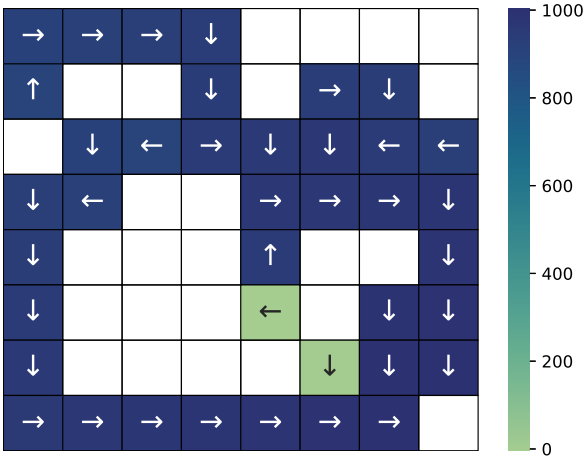

Fig. 15: Política *greedy* aprendida no tabuleiro 8 x 8 com chão não escorregadio.

nota-se a grande dificuldade da passagem em (1, 3) que não possui direção segura. Mesmo assim, a política aprendida em 8 é sub-ótima, já que teria menor chance de quedas se escolhe-se a esquerda ou a direita em vez de ir para baixo. Além disso, há múltiplas posições adjacentes a dois buracos, o que explica a baixíssima taxa de sucesso.

Finalmente, nota-se que, por 15, há muitas posições não visitadas com chão não escorregadio. Muitas dessas posições possuem valor entre os mais baixos (verdes) segundo a tabela Q obtida com chão escorregadio em 8. Desse modo, vê-se que o sistema de punições e recompensas concebido gera convergência rápida da política exploratória, mesmo com um ε mínimo de 0.01 e uma taxa de decaimento de 0.99, para os problemas com chão não escorregadio.

Em última análise, consideram-se os resultados obtidos com um tabuleiro 30 x 30 com chão não escorregadio.

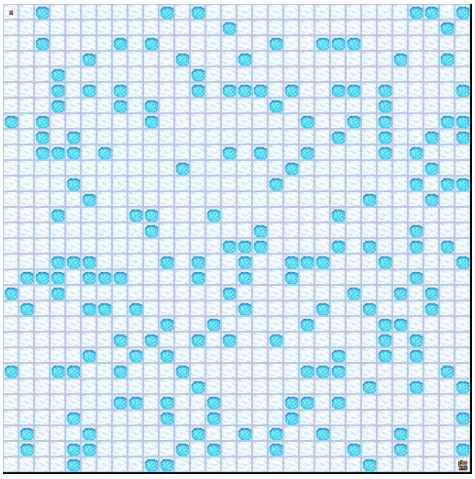

Fig. 16: Foto do tabuleiro 30 x 30 considerado.

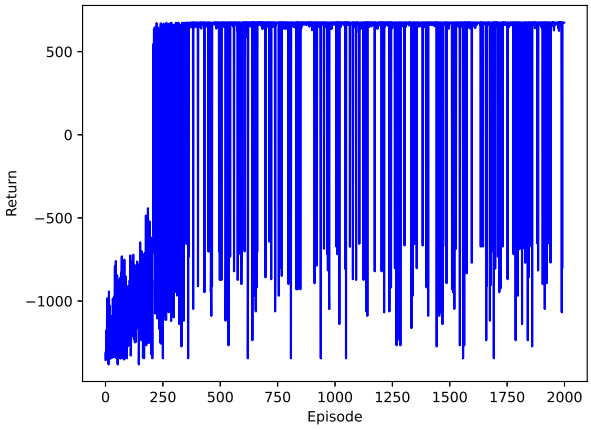

Fig. 17: Histórico de retorno no treino para o tabuleiro 30x30 com chão não escorregadio.

Enquanto 17 e 18 revelam rápida convergência da política exploratória, nota-se que sub-otimalidade da solução que possui um caminho mais longo do que o mínimo para o mapa em 16. Portanto, para o problema sem chão escorregadio o

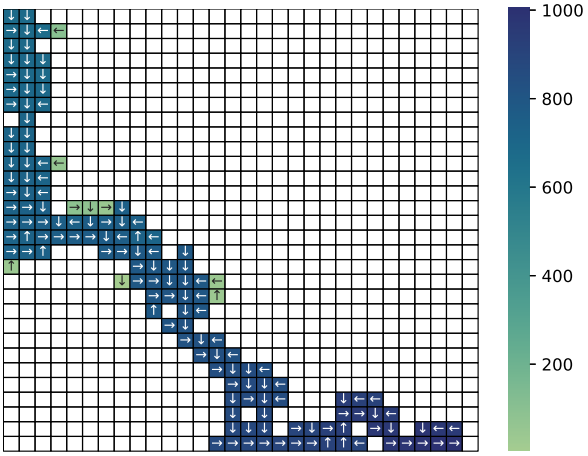

Fig. 18: Política *greedy* aprendida no tabuleiro 8 x 8 com chão não escorregadio.

sistema de punições e recompensas poderia ser melhorado, no entanto, isso provavelmente resultaria em um sistema que gera desempenho inferior para os problemas de chão escorregadio.

Mesmo assim, o fato de ter-se obtido soluções para mapas inteiramente diferentes para tamanhos de 4 x 4 até 30 x 30 indica que o algoritmo de *Q-Learning* aplicado tem o potencial de obter soluções para o problema geral do *Frozen Lake* com chão **não escorregadio**, dado que há espaço para melhorias no sistema de punições e recompensas, ao menos, se desconsiderada a limitação de espaço exigido para armazenamento da tabela Q, o que faria com que, para problemas suficientemente grandes, fosse necessário mudar sua forma de representação *e.g.* uma rede neural, como no *Deep Q-Learning* original.

IV. Conclusão

No presente trabalho aplicou-se uma forma de *Q-Learning* modificado que permitiu, com a escolha de pesos específicos para realizar *reward engineering*, obter resultados satisfatórios no problema do *Frozen Lake* do *Gymnasium*.

Constatou-se a dificuldade de se obter punições e recompensas com pesos adequados para o problema geral, enquanto existe a possibilidade de se melhorar cada solução utilizando-se valores otimizados para problemas particulares.

Finalmente, verifica-se a adequação do algoritmo *Q-Learning* ao problema, pois esse possui espaços de estados e de ações suficientemente pequenos, ao menos para o intervalo de tamanhos de tabuleiro considerado, e os resultados demonstraram ser possível obter soluções para a versão determinística do problema, sem a necessidade de se alterar os pesos das punições e recompensas, enquanto o problema não determinístico, por sua grande estocasticidade apresenta maiores dificuldades.