Pac-Maze

Eduardo Vieira e Sousa
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
eduardoatrcomp@gmail.com

***Abstract*—Este relatório aborda a aplicação do aprendizado por reforço a uma versão simplificada do jogo Pac-Man, onde o objetivo do jogador é caminhar por um labirinto em busca de uma "pílula", desviando dos fantasmas, que nessa versão do *game* não se movem. O algoritmo implementado no aprendizado foi o Q-Learning, que em seguida foi testado utilizando variadas combinações de valores para a taxa de aprendizado e o fator de exploração. Os testes foram realizados utilizando três mapas com diferentes tamanhos e níveis de dificuldade, e seus resultados foram compilados no formato de tabelas e gráficos.**

I. Introdução

O objetivo deste trabalho foi a implementação de um agente capaz de jogar uma versão simplificada do famoso *game* Pac-Man. Nessa versão, apelidada de **Pac-Maze**, o personagem deve se deslocar através do mapa em busca de uma única "pílula", que ao ser comida dá a vitória ao jogador, finalizando a partida. Durante o percurso, o personagem deve desviar dos fantasmas, que ficam estáticos no mapa, e caso entre em contato com algum deles, o jogador perde a partida.

II. Reinforcement Learning

Para "aprender" a jogar e conseguir completar o *game*, o agente utiliza uma técnica conhecida como aprendizado por reforço (**Reinforcement Learning**). De uma maneira simplificada, o método consiste em fazer com que o agente explore o ambiente, oferecendo a cada ação, recompensas positivas e negativas, com a finalidade de obter uma política que possa guiar a execução da tarefa de uma maneira desejada.

A. Q-Learning

Dentro do aprendizado por reforço existem os mais variados tipos de abordagens. Nesse trabalho, será utilizada uma técnica de *Temporal-Difference Learning* (**TD Learning**), conhecida como **Q-Learning**. Esse método consiste em definir uma Função Ação-Valor (**Q(s,a)**), geralmente representada por uma tabela, que guarda um valor para cada ação possível em todos os estados. De modo que, a medida que o agente interage com o ambiente, esses valores são atualizados de acordo com a diferença entre o valor atual e a recompensa recebida, somada a uma estimativa de recompensas futuras.

III. Modelagem

O **Pac-Maze** pode ser modelado como um Markov Decision Process (**MDP**), onde o conjunto de estados é composto de todas as posições do mapa onde o agente pode estar. Já as ações possíveis são os movimentos nas quatro direções: **Up (U)**, **Down (D)**, **Left (L)** e **Right (R)**. A função de recompensa dá ao agente 10 pontos ao encontrar a "pílula", -10 ao tocar um fantasma e -1 para cada movimento nos espaços vazios do mapa. E por fim, a função de transição é determinística, fazendo com que a ação tomada pelo agente sempre leve ao estado desejado.

A. Estado

O labirinto do game consiste em um plano, assim sendo, todas as suas posições podem ser representadas por um par ordenado **(x,y)**. Deste modo, os estados são implementados pela classe **Estado**, que contem um par valores inteiros representando as coordenadas do mapa, da seguinte maneira:

- **X**: um valor inteiro no intervalo entre 0 e o número total de linhas, indicando a linha onde o quadrante está localizado;
- **Y**: um valor inteiro localizado no intervalo entre 0 e o número total de colunas, indicando a coluna onde o quadrante está localizado.

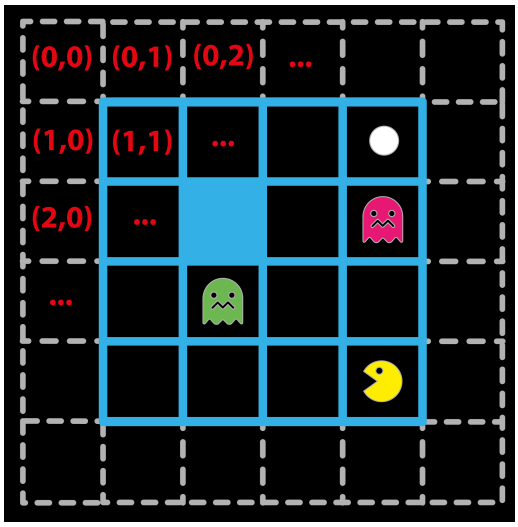

Fig. 1. Modelagem dos estados do jogo **Pac-Maze**

B. Agente

O agente é representado por uma classe que fica responsável por escolher a ação durante cada jogada, a fim de definir uma política para aquele mapa específico. Após executar cada ação, o agente recebe uma recompensa e atualiza a sua

Função Valor-Ação (**Q(s,a)**) para aquele estado do mapa, que está contido em uma tabela conhecida como **Q-Table**. Desta maneira, o agente vai "aprendendo" a medida que interage com o ambiente. Além disso, é utilizada a estratégia de exploração $\epsilon - Greedy$, para definir a proporção das ações que serão de exploração e de aproveitamento. A classe que implementa o agente possui os seguintes atributos:

- **fatorDesconto**: um valor real que tem a função de diminuir a importância de recompensas futuras, ajudando a priorizar recompensas mais imediatas;
- **taxaAprendizado**: utilizada para definir o "ritmo" de aprendizagem do agente. Um valor muito alto pode dificultar a convergência, enquanto um valor muito baixo pode fazer com que o treinamento demore muito tempo;
- **fatorExploracao**: define a porcentagem das ações que serão de exploração e de aproveitamento na estratégia E-greedy adotada pelo agente;
- **qTable**: matriz tridimensional $(Linhas \times Colunas \times Acoes)$ que guarda o **Valor-Ação** para todos os estados do mapa. Deste modo, para cada estado representado pela tupla $(Linha \times Coluna)$, temos quatro valores, um para cada uma das ações possíveis, que são representadas por **0** (R), **1** (L), **2** (U) e **3** (D), como mostra a figura (2).

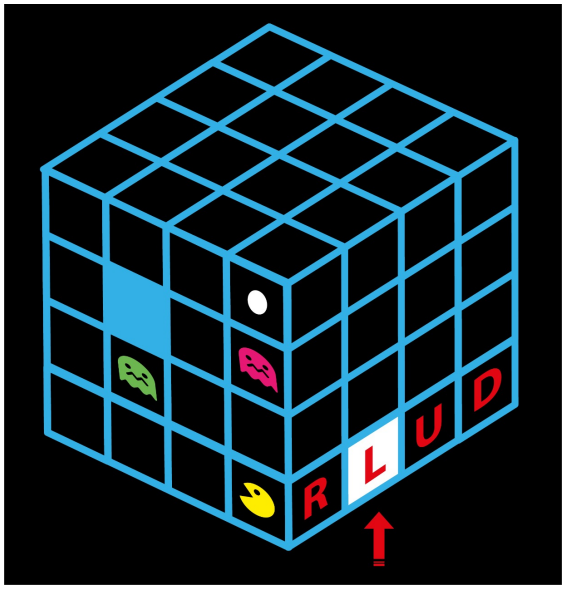

Fig. 2. Representação tridimensional da **Q-Table** implementada no agente

Como mencionado anteriormente, o agente fica encarregado de duas atividades principais que estão implementadas nos seguintes métodos:

- **getAcao(Estado)**: recebe um estado e retorna a ação escolhida pelo agente, que tem como base a estratégia $\epsilon - Greedy$. Isso é feito sorteando-se um número, e caso ele seja menor que o valor de ϵ, é escolhida uma ação aleatória, caso contrário, o agente escolhe a ação com maior valor de sua **Q-Table** para o estado dado;
- **aprender(Estado anterior, Estado Atual, Ação, Recompensa)**: esse método tem a função de atualizar a Função Estado-Valor do agente, representada pela **Q-Table**. Isso é feito utilizando a seguinte equação (1);

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma MaxQ(s',a') - Q(s,a)] \quad (1)$$

- **maiorValor(x, y)**: recebe dois inteiros que representam as coordenadas de um estado e retorna a ação com maior valor associada a ele.

C. Ambiente

O mundo é implementado na classe **Ambiente**, que tem a função de aplicar as ações escolhidas pelo agente sobre mapa e devolver as recompensas. Os atributos contidos na classe são os seguintes:

- **numLinhas**: um inteiro que representa o número de linhas do mapa;
- **numColunas**: inteiro que representa o número de colunas do mapa;
- **mapa**: matriz de caracteres que contém as informações relacionadas a cada estado do mapa.

Os métodos implementados na classe foram:

- **mover(Estado, Ação)**: recebe um **Estado** e um inteiro, que representa a ação escolhida, devolvendo o estado resultante da aplicação dessa ação. Para isso, o método checa se é possível executar a ação, ou seja, se não existe uma parede na direção desejada. Caso seja possível, o novo estado é retornado, caso o contrário o esto atual é retornado;
- **funcaoRecompensa(Estado)**: recebe um **Estado** e retorna a recompensa por estar naquele estado;
- **isFinal(Estado)**: recebe um **Estado** e verifica se este é um estado final, ou seja, se contém um fantasma ou uma "pílula";
- **reinsereAgente()**: retorna um **Estado** vazio, ou seja, estados que não contém fantasmas, "pílulas" ou paredes. Deste modo, podendo reiniciar a posição do agente para um novo episódio de treino.

D. PacMaze

Por fim, temos a classe principal, que recebe, em ordem, os seguintes parâmetros:

1) Uma *string* que contém o caminho para o arquivo contendo **numLinhas**, **numColunas** e os caracteres que representam o labirinto para construção do modelo;
2) Um número real contendo a **taxaAprendizado** para o treinamento do agente;
3) Um real contendo o **fatorExploracao** para a definição da estratégia $\epsilon - Greedy$;
4) Um inteiro que define quantos episódios deve durar o treinamento do agente.

Desta maneira, a classe principal fica encarregada da leitura dos dados de entrada, do treinamento do agente e da gravação dos dados de saída em um arquivo. A dinâmica de treinamento, por sua vez, consiste em inserir o **Agente** em uma das posições possíveis do mapa, depois enviar à classe **Ambiente** o seu

Estado atual, juntamente com a ação escolhida pela estratégia $\epsilon - Greedy$.

Em seguida, o **Ambiente** deve retornar o **Estado** atualizado após a aplicação da ação, juntamente com a sua recompensa. O agente atualiza sua **Q-Table**, e o processo recomeça, repetindo-se até que o número de episódios desejado seja atingido. Esse processo está representado na figura (3).

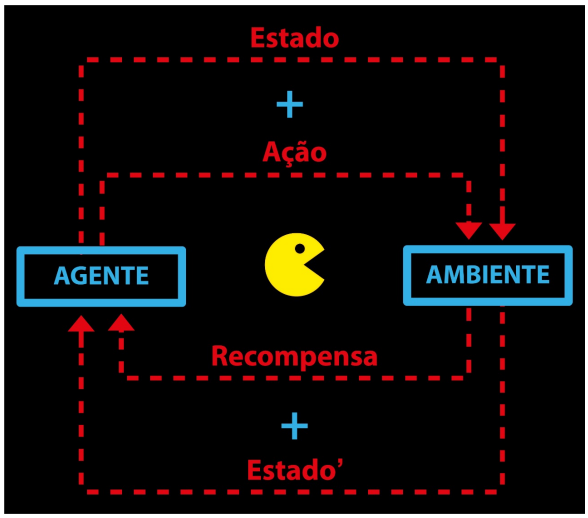

Fig. 3. Dinâmica de treinamento do agente

E. Entrada e Saída

O arquivo de entrada contém em sua primeira linha dois inteiros: N indicando a quantidade de linhas e M a quantidade de colunas. Nas próximas N linhas, cada uma contendo M caracteres, está representada a estrutura do mapa onde as paredes do labirinto são representadas pelo caractere **"#"**, já os espaços vazios são definidos por **"-"**. Os fantasmas aparecem na matriz como o símbolo **"&"**, e por fim, a "pílula" é representada por **"0"**. Na figura (3), está exemplificado uma entrada e sua modelagem correspondente.

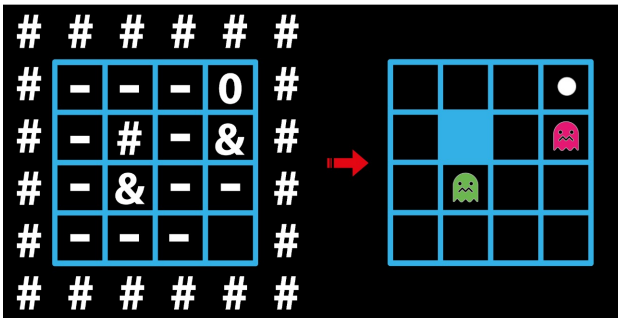

Fig. 4. Exemplo da conversão de um artigo de entrada

Após a execução do programa, são gerados dois aquivos de saída. O primeiro, **"q.txt"** contém uma lista de **Valor-Ação** para todos os estados do mapa. Cada linha possui dois inteiros, separados por vírgula, que indicam as coordenadas do estado. Em seguida temos a letra que representa a ação (**R, L, U, D**), seguida do seu valor correspondente, representado por um real com precisão de três casas decimais.

Por fim, o segundo arquivo gerado, **"pi.txt"**, contém a matriz de caracteres do arquivo de entrada, onde cada estado vazio é substituído por sua ação com o maior valor. Deste modo, a saída corresponde à política gerada pelo treinamento para o mapa dado.

IV. Políticas

A seguir estão apresentados os resultados dos arquivos **"pi.txt"**, contendo as políticas geradas para as três entradas fornecida1s.

A. pacmaze-01-tiny

```
##############
#RRRRR0LLLLLL#
#######U##U#U#
#RRRRRRU&RU#U#
#U##########&#
#ULLLLLLLLLLL#
##############
```

B. pacmaze-02-mid-sparse

```
####################
#DLLLLLLLLLLLLL#RRD#
#DLL&DL&DL&DLLL#RRD#
#DLLLLLLLLLLLLL#U#D#
#D##############U#D#
#RRRRRRRRRRRRRRRU#D#
##################D#
#DLLLLLLL&RRRDLLLLL#
#D&DL########DLL&DL#
#DLLLLLLLLLLLLLLLLL#
#D################U#
#D#RRRRD#0#RRRRRRRU#
#RRRU#RRRU#RRRRRRRU#
####################
```

C. pacmaze-03-tricky

```
####################
#DLLL&RRRDLLLL&RRRD#
#D&&&&&&&&D&&&&&&&&D#
#RRRRRRD&D&DLLLLLLL#
#RRD&RDD&D&DLL&DLLL#
#RRRRRRD&D&DLLLLLLL#
#RRRRU&RRDLL&ULLLLL#
#RRRRRRU&D&ULLLLLLL#
#RRU&RUU&D&ULL&ULLL#
#RRRRRRU&0&ULLLLLLL#
####################
```

D. Observações

Existem estados onde uma ação se mostra claramente superior às demais, como por exemplo um caminho entre um corredor de fantasmas, fazendo com que a melhor política seja

uma linha reta dentro deste caminho na direção da "pílula". No entanto, em muitos casos, não existe uma ação objetivamente melhor, fazendo com que duas ações possuam o maior **Valor-Ação** para aquele estado. Caso isso ocorra, a política exibida no arquivo **"pi.txt"** pode variar um pouco, dependendo da implementação do algoritmo que encontra o maior **Valor-Ação** dado um estado. No entanto, isso não altera o resultado final, fazendo com que na prática, ambas as políticas sejam iguais.

V. Testes e Resultados

Para a realização dos testes, os três arquivos de entrada fornecidos foram utilizados variando-se os parâmetros α e ϵ entre 0.1, 0.3, 0.6 e 0.9, durante seções de 300 episódios para cada combinação. Em seguida mediu-se a recompensa por episódio, chegando por fim à recompensa média, dividindo esse valor pelo número total de episódios.

Começando pelo arquivo **"pacmaze-01-tiny"**, fica fácil reparar que o aumento na taxa de aprendizado trouxe uma melhora na recompensa para os valores de ϵ de 0.1 até 0.6. Porém com um fator de exploração muito alto, a taxa de aprendizado igualmente alta só dificultou a convergência, piorando as recompensas, como mostra a tabela (I).

TABLE I
Resultados do treinamento para **"pacmaze-01-tiny"**

	α			
ϵ	**0.1**	**0.3**	**0.6**	**0.9**
0.1	-19.55	-7.02	-3.77	-2.95
0.3	-21.75	-10.15	-8.72	-5.83
0.6	-28.91	-18.06	-16.10	-15.41
0.9	-46.07	-52.26	-55.95	-61.02

Observando o gráfico (5), fica facilmente perceptível que o aumento no ϵ gera uma diminuição na recompensa média. Isso ocorre pois o mapa é pequeno, logo ele não necessita de muita exploração para se encontrar uma boa política. Essa diminuição ocorre especialmente nos testes com alta taxa de aprendizagem, combinação que acaba dificultando na convergência. Para esta entrada, as melhores recompensas foram obtidas com $\epsilon = 0.1$ e $\alpha = 0.9$.

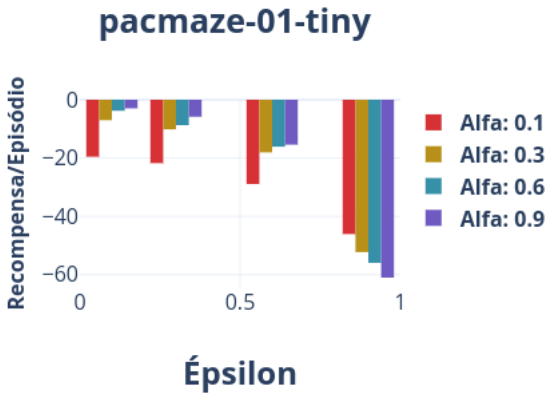

Fig. 5. Gráfico do treinamento para **"pacmaze-01-tiny"**

Já no mapa **"pacmaze-02-mid-sparse"**, o aumento da taxa de aprendizagem ajudou em todos os casos, como indicado na tabela (II). Isso se deve ao tamanho do mapa, que por ser maior, faz com que uma maior taxa de aprendizado seja mais útil, já que não será possível explorar o mapa como um todo tão bem quanto os demais mapas, realizando a mesma quantidade de episódios em cada treinamento.

TABLE II
Resultados do treinamento para **"pacmaze-02-mid-sparse"**

	α			
ϵ	**0.1**	**0.3**	**0.6**	**0.9**
0.1	-186.88	-145.05	-109.08	-92.21
0.3	-118.59	-94.99	-86.74	-71.11
0.6	-85.53	-65.98	-69.72	-59.82
0.9	-96.62	-80.40	-76.86	-74.63

O gráfico (6) mostra que o aumento no ϵ proporcionou melhoras nas recompensas até o valor de 0.6. Sendo que a partir daí, aparentemente o aumento começa a prejudicar as recompensas do agente. Neste caso, os melhores resultados foram obtidos com os parâmetros $\epsilon = 0.6$ e $\alpha = 0.9$.

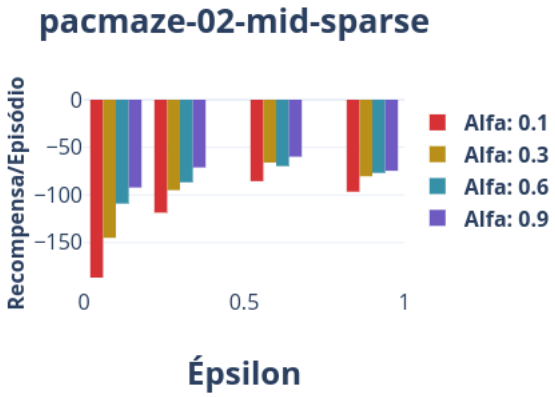

Fig. 6. Gráfico do treinamento para **"pacmaze-02-mid-sparse"**

Por fim, o mapa **"pacmaze-03-tricky"**, que possui a maior quantidade de fantasmas, não exibe grande melhora na recompensa média proveniente do aumento de ϵ e α. Isso ocorre pois esse é o ambiente mais difícil, resultando em muitas mortes para o agente e dificultando a convergência do método e consequentemente o aprendizado, como mostra a tabela (III).

TABLE III
Resultados do treinamento para **"pacmaze-03-tricky"**

	α			
ϵ	**0.1**	**0.3**	**0.6**	**0.9**
0.1	-46.69	-30.62	21.91	-19.38
0.3	-31.78	-28.24	-23.62	-23.73
0.6	-25.55	-22.78	-21.14	-20.78
0.9	-20.67	-22.62	-22.19	-21.77

Para este mapa, os parâmetros que obtiveram as maiores recompensas foram $\epsilon = 0.9$ e $\alpha = 0.1$, porém em todos os casos as recompensas foram muito próximas, mostrando a

menor variação entre todos os testes, mesmo com altas taxas de exploração, e variando-se o α como mostra o gráfico (6). Isso confirma que o método não conseguiu atingir convergência. Deste modo, uma quantidade maior de episódios seria necessária para que o agente pudesse desenvolver uma política ótima nesse ambiente.

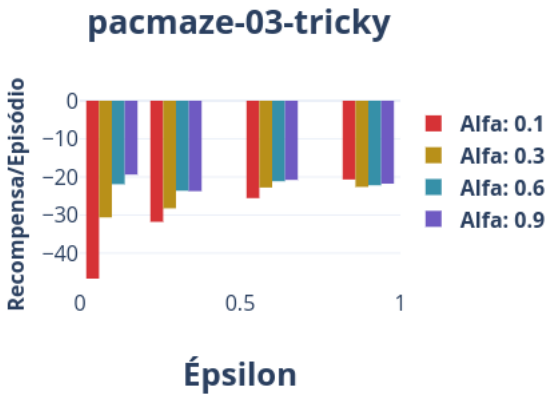

Fig. 7. Gráfico do treinamento para **"pacmaze-03-tricky"**

VI. Considerações Finais

Os experimentos demonstraram que o **Q-Learning** é um algoritmo extremamente eficiente na solução do **Pac-Maze**. Com o número adequado de episódios durante o treinamento, ele é capaz de resolver praticamente qualquer tamanho de labirinto, mesmo utilizando uma tabela para representar sua **Função Ação-Valor**.

Entretanto, caso os fantasmas se movessem, possivelmente a matriz não seria suficiente para representar a quantidade de estados, sendo necessária a utilização de *features* para descrever os estados e aproximação de função no lugar da **Q-Table**.

References

[1] Sutton, Richard S. and Barto, Andrew G. Reinforcement Learning: An Introduction (2rd ed.). MIT Press, Cambridge, MA, 2018.