

Trabalho Prático 2 - Introdução à Inteligência Artificial

Francisco Teixeira Rocha Aragão - 2021031726

Data: 02 de fevereiro de 2025

1 Introdução

O presente trabalho busca exercitar os conhecimentos sobre aprendizado por reforço ao aplicar o algoritmo Q-Learning e suas variações no contexto do problema de "Path Finder", comumente aplicados em jogos ao se encontrar caminhos que levam ao objetivo desejado. Desse modo, o trabalho busca aprender a política (mapa com os caminhos a serem seguidos) que leva ao objetivo final, maximizando a recompensa obtida.

2 Descrição do problema e implementação

O problema foi definido como um tabuleiro de dimensões $W \times H$ (largura x altura) com um agente que se move em 4 direções (cima, baixo, esquerda e direita) e que deve encontrar o caminho que leva ao objetivo final. O agente pode se mover para qualquer direção, mas não pode sair do tabuleiro nem passar pelas paredes. O mapa é composto por diferentes tipos de terrenos (no caso, cada terreno representa um possível estado), cada um com sua própria recompensa. O objetivo é encontrar o caminho que leva ao objetivo final, maximizando a recompensa obtida. Cada tipo de algoritmo implementado (comentado abaixo) possui recompensas associadas a cada um dos terrenos, ou movimentações específicas que lidam com probabilidades, influenciando o tipo de movimentação do agente.

3 Algoritmos implementados

A ideia do trabalho é implementar o Q-Learning e suas variações, para então comparar os resultados em diferentes aplicações. O algoritmo Q-Learning funciona de forma iterativa, em que é definida a função $Q(s, a)$ que representa a recompensa descontada esperada ao se tomar a ação A no estado S seguindo alguma política. Desse modo, a cada nova movimentação realizada, a função Q é atualizada para essa ação e esse estado, e assim o processo continua por um número de passos, retornando ao fim a política aprendida.

Desse modo, foi-se implementado a tabela da função Q como definida abaixo, em que temos uma matriz de dimensões (altura x largura), e para cada posição, temos um vetor de 4 dimensões (cima, baixo, esquerda, direita) que representa a recompensa esperada para cada uma dessas ações para cada estado do mapa (posições no tabuleiro).

```
# código utilizado para inicialização da tabela Q
self.Q = np.zeros((height, width, len(self.directions)))
```

Com isso, a execução do algoritmo é baseada nos parâmetros α que controla a taxa de aprendizado, γ que controla o fator de desconto de ações futuras, e ϵ que controla a probabilidade o balanço entre exploração x exploração. Todas essas variáveis estão definidas na formulação abaixo:

$$Q(s, a) = Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)) \quad (1)$$

Desse modo, três variações do algoritmo foram implementadas:

3.1 Standard Q-Learning

O algoritmo padrão de Q-Learning foi implementado com a função de recompensa definida como:

- Grama: -0.1
- Grama Alta: -0.3
- Água: -1
- Fogo: -10
- Objetivo: 10
- Parede: -infinito

Os parâmetros foram $\alpha = 0.1$, $\gamma = 0.9$, $\epsilon = 0.1$.

3.2 Positive Q-Learning

O algoritmo Positive Q-Learning foi implementado com a função de recompensa definida como:

- Grama: 3
- Grama Alta: 1.5
- Água: 1
- Fogo: 0
- Objetivo: 10
- Parede: -infinito

Os mesmos parâmetros do algoritmo padrão foram utilizados.

3.3 Stochastic Q-Learning

O algoritmo Stochastic Q-Learning foi implementado com os mesmos parâmetros e recompensas do algoritmo padrão. No entanto, agora existem probabilidades associadas a cada movimento. Ao se executar uma ação, existe 80% de chance de se mover na direção desejada, e 20% de chance de mover-se nas direções adjacentes. Ou seja, se vou para cima, tenho 80% de chance de ir para cima, e 20% de chance de ir para a esquerda ou direita.

4 Execução

Para executar o código, basta seguir:

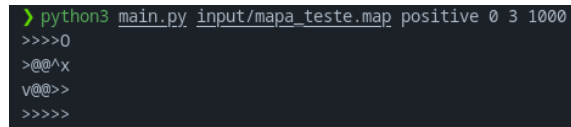
```
python3 main.py <caminho_input> <metodo> <inicio_x> <inicio_y> <iterações>
```

```
# método: standard, positive, stochastic  
# exemplo: python3 main.py input.map standard 0 0 1000
```

5 Resultados e Análises

Testes feitos em uma máquina com i5 de décima primeira geração, 16gb de ram e Debian 12.

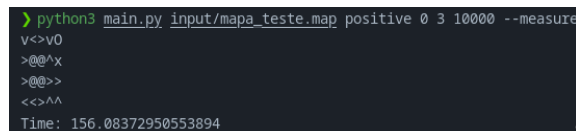
Inicialmente, cabe destacar os resultados obtidos para a variação 'positive' do algoritmo Q-Learning. Como essa variação não possui recompensas negativas (ou seja, não temos punições para caminhos ruins), não existe a necessidade de focar-se na exploração do ambiente para encontrar o caminho correto ao objetivo. Com isso, comportamentos 'estranhos' acontecem nesse caso com o algoritmo não finalizando em muitas iterações (pelo menos não em tempo hábil), ou então retornando políticas contra intuitivas. Como exemplo, temos a imagem 1 disponível abaixo em que a política retornada após 1000 iterações prefere em alguns momentos andar em direção a parede (ou seja, continua acumulando recompensas ao longo das iterações).



```
> python3 main.py input/mape_teste.map positive 0 3 1000  
>>>>0  
>@@^x  
v@@>>  
>>>>
```

Figure 1: Positive - Mapa simples - 1.000 iterações

Vale destacar que ao aumentar-se o número de iterações, o tempo da variação 'positive' torna-se muito grande como visto na imagem 2. Na menor instância de testes, com 10.000 iterações o algoritmo apresentou um tempo muito pior em comparação aos demais (será mostrado mais abaixo), não sendo justificado esse tempo para uma instância tão pequena. Desse modo, outros testes não foram executados para esse método.



```
> python3 main.py input/mape_teste.map positive 0 3 10000 --measure  
v<>v0  
>@@^x  
>@@>>  
<<>^^  
Time: 156.08372950553894
```

Figure 2: Positive - Mapa simples - 10.000 iterações

Para a variação 'stochastic', assim como o método 'positive', temos comportamentos estranhos a primeira vista, porém que são justificáveis ao observa-se as probabilidades associadas as movimentações feitas pelo agente. Na imagem 3 mostrada abaixo, embora a política faça sentido na prática, começamos a perceber o desempenho do algoritmo em comparação ao método 'standard' (mostrado mais abaixo), possuindo um tempo pior em sua execução.

```
~/Documents/Francisco/Faculdade/7_Semestre/introducao_IA/tp2/q_learning
> python3 main.py input/mapa_teste.map stochastic 0 3 100000 --measure
>>>Q
^@^x
^@^v
^>>^<
Time: 4.523851156234741
```

Figure 3: Stochastic - Mapa simples - 100.000 iterações

Já no caso da imagem 4, com o mapa 'maze' em que temos um caminho longo e outro curto ao objetivo, a variação 'stochastic' apresenta comportamentos incomuns. O caminho mais curto que fica presente na lateral esquerda do mapa, apresenta alguns movimentos em direção a parede. Isso pode parecer estranho a primeira vista, porém faz todo sentido prático. Como existem probabilidades de mover-se em direções adjacentes, em alguns movimentos para baixo (direto ao objetivo) o método 'stochastic' pode acabar indo para a lateral e no pior caso caindo no fogo ('x'). Desse modo, ao mover-se para a parede, o risco de cair no fogo é retirado, já que os movimentos possíveis são bater na parede, ou então ir pra cima, ou para baixo (direção do objetivo). Desse modo, essa ação mais segura é aprendida pelo algoritmo e retornada na política. Esse mesmo comportamento é observável na imagem 5 com a direção 'oposta' ao fogo sendo escolhida ao invés de andar direto ao objetivo em alguns casos.

```
> python3 main.py input/maze.map stochastic 10 0 300000 --measure
x@x@x@x@x@v@
v^vvv<v>v@
<x@v@v@v@v@
<x@v<<<<<<@
vx@v@v@v@v@
vx@>>>>>><x
vx@v@v@v@v@
Q<<<<<<<<@
Time: 71.63173294067383
```

Figure 4: Stochastic - Labirinto - 300.000 iterações

```
> python3 main.py input/choices.map stochastic 5 0 300000 --measure
@>>v^<^<^@
^@<x@x>^@
@v@v@x>@v@
@v@vx@x>@v@
@>>v@x>>v@
@v@<x@xv@v@
@v@v@xv@v@
@>>>Q<<<<@
Time: 35.1125009059906
```

Figure 5: Stochastic - Múltiplos caminhos - 300.000 iterações

Já no caso da variação 'standard', temos um comportamento mais condizente com o esperado, sendo possível perceber um caminho que leva ao objetivo final. Isso é perceptível tanto em mapas pequenos quanto em grandes, como vistos nas imagens 6, 7 e 8 disponíveis abaixo. De modo geral percebe-se que esse é o método mais eficiente em relação ao tempo, possuindo execuções mais rápidas em comparação as outras variações comentadas acima. Para o resultado da imagem 6, temos que o caminho até a política foi encontrado gastando menos de 3 segundos para 100.000 iterações. Isso

mostra o contraste com a variação 'positive' apresentada acima que gastou mais de 150 segundos com 10.000 iterações. No caso da imagem 7, o mapa 'maze' possui um caminho longo e outro curto até o objetivo. Nesse exemplo, o método encontrou facilmente a rota mais curta, com a rota mais longa possuindo uma política não muito bem definida ao objetivo. Esse resultado também é válido visto que, como a rota curta é facilmente aprendida, a rota longa é pouco utilizada, logo, não é melhorada ao longo das iterações.

```
> python3 main.py input/mape_teste.map standard 0 3 100000 --measure
>>>0
v@@^x
v@@^<
>>>^^
Time: 2.858001232147217
```

Figure 6: Standard - Mapa simples - 100.000 iterações

```
> python3 main.py input/maze.map standard 10 0 300000 --measure
x@x@x@x@x@v@
v<<<<<<<<<<@
vx@000000000^@
vx@>^^v<>>^@
vx@^000000000
vx@<^>^v^^vx
vx@000000000@
0<<<<<<<<<<^@
Time: 19.390738248825073
```

Figure 7: Standard - Labirinto - 300.000 iterações

Vale comentar que no caso do mapa 'choices', observa-se como o método busca o caminho com menor punição (caminhos de grama), em contraste aos caminhos com água ou grama alta nas laterais do mapa, em que muitas vezes observa-se a política indo 'contra' essa rota, como visto na imagem 8 disponível abaixo. Assim como ocorreu no exemplo descrito acima, outras rotas possíveis piores do que o caminho mais curto são menos escolhidas ao longo do tempo, assim a política ótima não é muito bem definida.

```
> python3 main.py input/choices.map standard 5 0 300000 --measure
@>>v<<<<<<<@
^@vx@xv@>@
^@v@xv@>@
@v@vx@xv@>@
@>>v@xv<>@
^@vx@xv@>@
@v@v@xv@^@
@>>>0<<<<<@
Time: 13.098892211914062
```

Figure 8: Standard - Múltiplos caminhos - 300.000 iterações

6 Conclusão

O presente trabalho mostrou-se útil para melhorar o entendimento sobre Aprendizado por Reforço e principalmente sobre o Q-Learning, importante algoritmo para aprendizado de políticas em am-

bientes desconhecidos. Percebe-se então como pequenas mudanças nas regras do problema geram resultados tão diversos. A variação 'positive' mostrou-se ruim na prática, possuindo um desempenho em tempo muito pior em comparação aos demais, além de não apresentar movimentações coerentes com o jogo, o que é justificado pela falta de punições nas recompensas dos movimentos. A variação 'standard' (padrão) mostrou-se a melhor na prática, tendo bom tempo de execução quanto políticas condizentes com a entrada que levam ao objetivo final em um curto trajeto. Já a variação 'stochastic' mostrou-se interessante, em que mesmo não apresentando caminhos mais intuitivos a primeira vista, reforçou como o algoritmo Q-Learning é capaz de aprender políticas em ambientes com incertezas, focando em evitar a chance de executar movimentações ruins para chegar ao objetivo.