



Universidade Federal de Santa Catarina
Centro Tecnológico
DAS410058-41 - **Aprendizado de Máquina** - 2020/3
Departamento de Automação e Sistemas
Professor: Jomi F. Hübner

Lara Popov Zambiasi Bazzi Oberderfer

Data: 13/03/2021

Relatório: Aprendizado Indutivo - Pacman

Objetivo: O objetivo do trabalho é implementar e experimentar a resolução de problemas com um aprendizado por reforço Q-Learning (QL).

Problema 2: Implementar um agente pac-man que aprende por reforço. Sugere-se usar o material disponível em <https://inst.eecs.berkeley.edu/~cs188/sp20/project3/>, em especial a **questão 9**.

Relatório: Conforme cronograma da disciplina, deve ser entregue um relatório descrevendo a modelagem do problema (o MDP), a implementação da solução e o resultado dos experimentos.

Modelagem do problema (MDP - Processo Decisório de *Markov*)

MDP do SmallGrid:

	1	2	3	4	5	6	7
1	%	%	%	%	%	%	%
2	%	😊					%
3	%		%	%	%		%
4	%		%	★			%
5	%		%	%	%		%
6	%	★	👾				%
7	%	%	%	%	%	%	%

Figura 1: Labirinto smallGrid

Estados: todas as posições que o agente pode observar de estados no ambiente.

O conjunto $S = \{ S_{1,1}, S_{1,2}, \dots, S_{6,5}, S_{7,7} \}$

Ações: todas as ações que o agente pode tomar no ambiente.

O conjunto $A = \{ N, O, L, S, \text{Stop} \}$

Recompensas: modela a recompensa que o agente receberá quando recebe quando realiza uma ação no ambiente.

Definido em gridworld.py = recompensa(estado, ação, proximoEstado)

A recompensa obtida pelo Pacman corresponde à diferença na pontuação do jogo.

Se recompensa($S_{2,2}$, N, $S_{2,3}$) (recompensa positiva)

Se recompensa($S_{2,2}$, S, $S_{2,1}$) (recompensa negativa)

Para outros layouts como o mediumClassic a recompensa é um pouco mais complexa. Nessa configuração o Pac-man é recompensado com uma pequena recompensa positiva por comer pontos e uma alta positiva recompensa ao comer fantasmas assustados ou vencer. Uma perda (comida por fantasmas) resulta em um resultado negativo alto recompensa. Uma pequena recompensa negativa é dada ao longo do tempo, a fim de defender soluções rápidas.

Parâmetros das Recompensas:

Comer um fantasma: 200 (para os outros labirintos que contém a opção de comida para comer os fantasmas)

Comer uma comida: 10

Perder (o fantasma alcançar o pacman): - 500

Punição de tempo (cada passo decrementa tempo): -1

Ganhar: 500 (se todas as comidas acabares)

Transições:

$T\{S_{2,2}, N, S_{2,1}\} = 0$, $T\{S_{2,2}, O, S_{1,2}\} = 0$, $T\{S_{2,2}, S, S_{2,3}\} = 1$,
 $T\{S_{2,2}, L, S_{3,2}\} = 1$, ...

Início de jogo:

😊 Pacman na posição $S_{2,2}$

👾 Fantasma na posição $S_{6,3}$

★ Comida nas posições: $S_{6,2}$, $S_{4,4}$

Fim de jogo:

Quando 😊 Pacman “come” todas as ★Comidas

Quando 👾 Fantasma se encontra na mesma posição do 😊 Pacman

Representações:

% - Parede

😊 - Pacman

★ - Comida

👾 - Fantasma

Ajustando os parâmetros do QL:

1. O primeiro passo foi ajustar o agente de iteração de valor

Arquivos alterados: valueIterationAgents.py (agente de iteração de valor)

Sendo a iteração de valor um método para calcular a política e valor de MDP ideal, o algoritmo de iteração de valor para resolver trabalhos de MDP resolvendo iterativamente as equações relacionadas à utilidade de cada estado para aqueles de seus vizinhos. Depois de aplicar a iteração de valor em nosso mundo da grade, obtemos os seguintes valores para cada estado:



Figura 2: Valor depois de 1 iterações



Figura 3: Valor depois de 10 iterações

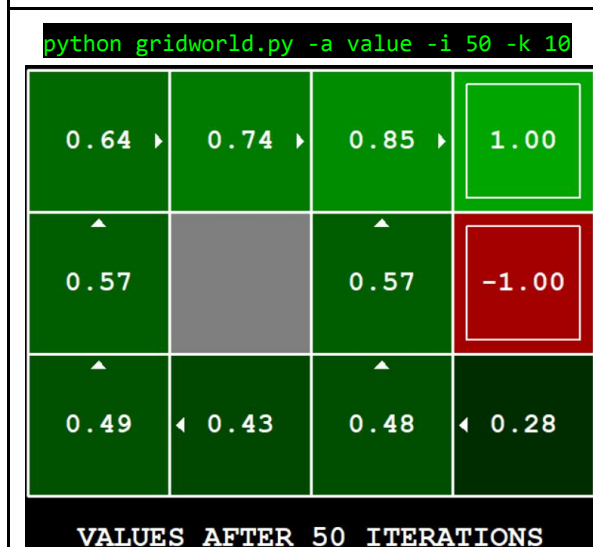


Figura 4: Valor depois de 50 iterações

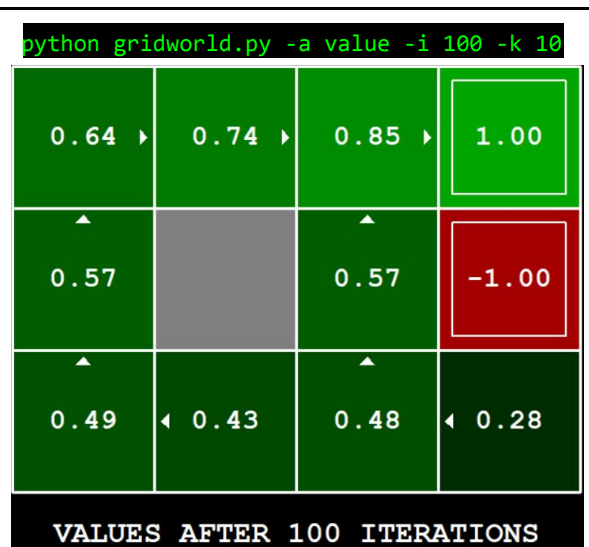


Figura 5: Valor depois de 100 iterações

2. O segundo passo foi ajustar o Cruzamento de Ponte (BridgeGrid)

Arquivos alterados: analisys.py

A BridgeGrid é um mapa em grade com um grande estado terminal de baixa e de alta recompensa separados por uma ponte estreita. Alterando os parâmetros fornecidos com o objetivo dos agentes entrarem dos terminais de baixa recompensa várias vezes e aprenderem que são ruins e tentar novas opções para escolher o melhor caminho, que é o da recompensa mais alta.

```
python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.2
```

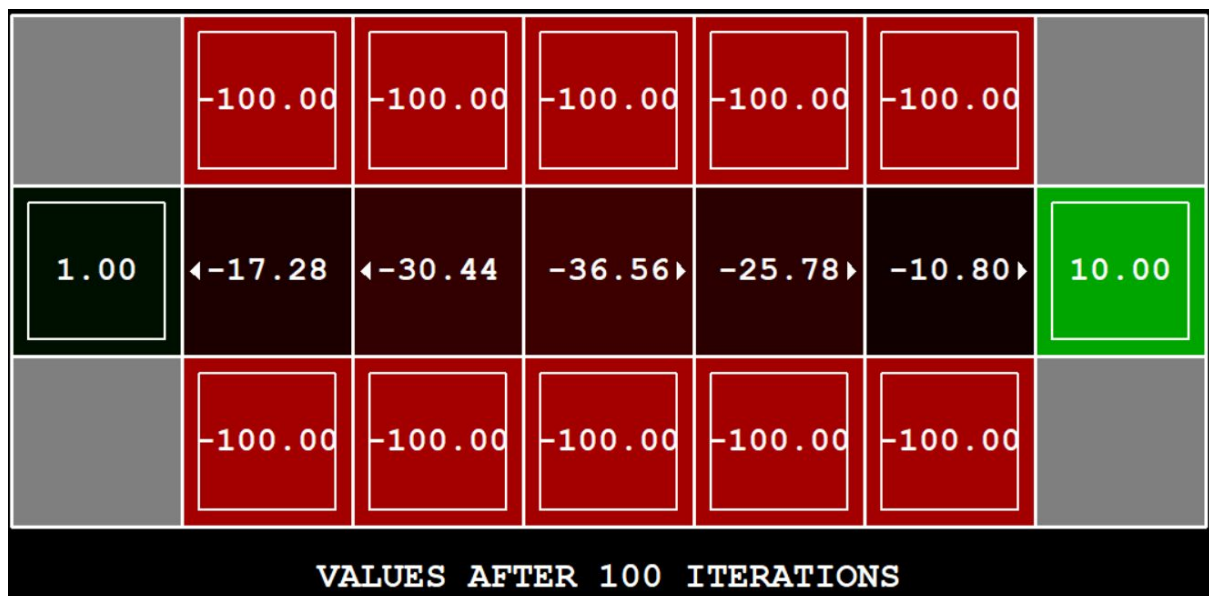


Figura 6: Mapa de estados depois de 100 iterações

3. O terceiro passo foi alterar os parâmetros para testar as Políticas (Questão 3)

Arquivos alterados: analisys.py

É uma distribuição de probabilidade sobre as ações dadas em um estado. É o comportamento do agente ou como ele escolhe suas ações em algum estado específico.

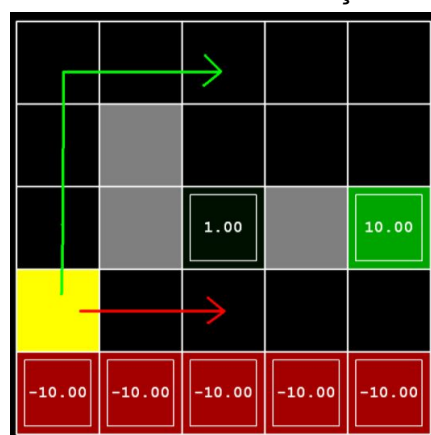


Figura 7: Mapa de desconto

Considere o DiscountGrid layout. Esta grade tem dois estados terminais com payoff positivo (na linha do meio), uma saída próxima com payoff +1 e uma saída distante com payoff +10.

A linha inferior da grade consiste em estados terminais com retorno negativo (mostrado em vermelho); cada estado nesta região de "penhasco" tem retorno de -10.

O estado inicial é o quadrado amarelo.

Podemos distinguir entre dois tipos de caminhos:

(1) caminhos que "arriscam o penhasco" e viajam perto da linha inferior da grade; esses caminhos são mais curtos, mas correm o risco de gerar um grande retorno negativo, e são representados pela seta vermelha na figura abaixo.

(2) caminhos que "evitam o penhasco" e viajam ao longo da borda superior da grade. Esses caminhos são mais longos, mas têm menos probabilidade de gerar grandes resultados negativos.

Aqui estão os tipos de política ideais que você deve tentar produzir:

Prefira a saída fechada (+1), arriscando o penhasco (-10)

Prefira a saída fechada (+1), mas evitando o penhasco (-10)

Prefira a saída distante (+10), arriscando o penhasco (-10)

Prefira a saída distante (+10), evitando o precipício (-10)

Evite as saídas e o penhasco (portanto, um episódio nunca deve terminar)

Para verificar suas respostas, execute o autograder:

```
python autograder.py -q q3
```

question3a() através de question3e() cada um deve retornar uma tupla de 3 itens de (desconto, ruído, recompensa vital) em analysis.py.

Melhores parâmetros que encontrei para a questão:

```
answerDiscount = 0.95      # desconto
answerNoise = 0            # ruído
answerLivingReward = 1000  # recompensa vital
```



Figura 8: Mapa de descontos depois de 100 iterações

4. Iteração de valor assíncrono

Arquivos alterados: valuelterationAgents.py

Após completar o agente de iteração de valor, com a equação do valor de iteração:

$$V_{k+1}(s) \leftarrow \max_{uma} \sum_{s'} T(s, uma, s') [R(s, uma, s') + \gamma V_k(s')]$$

Realizar a simulação: **python gridworld.py -a asynchvalue -i 1000 -k 10**

O simulador irá computar a seguinte política:

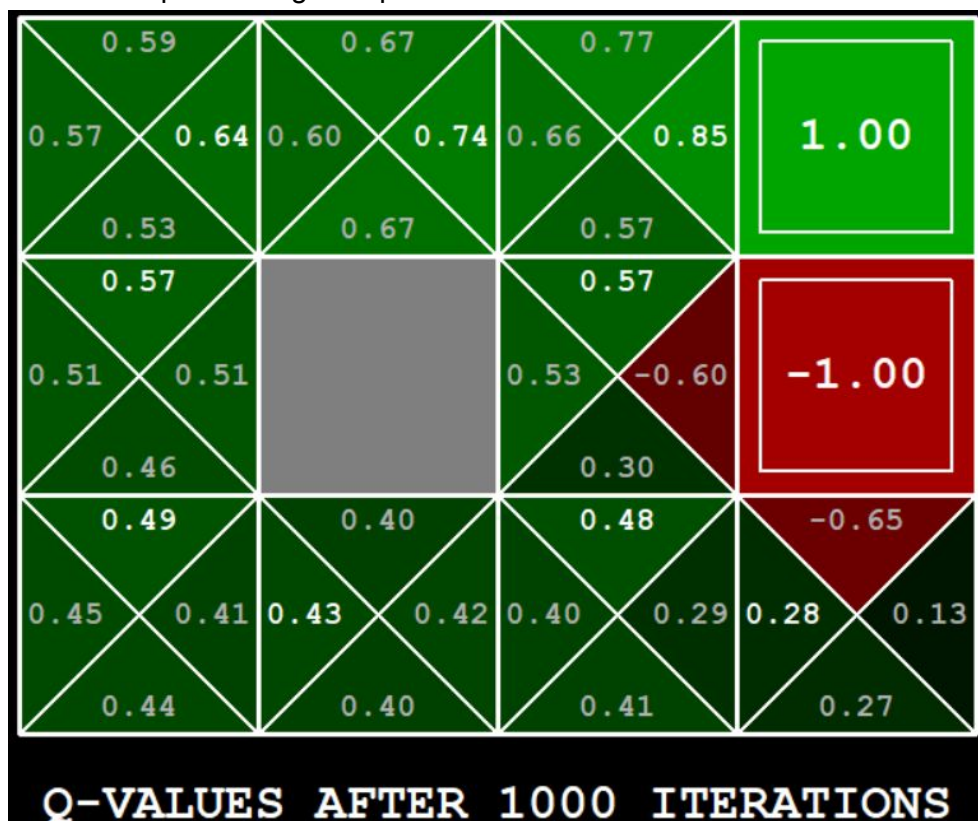


Figura 9: Mapa de políticas depois de 1000 iterações

Depois de percorrer os valores e a simulação, o agente encontra esse melhor valor de iteração para cada posição da grade:



Figura 10: Mapa de políticas depois da simulação de 1000 iterações

5. Q-Learning e Pacman (Questão 9)

O Reinforcement Learning (Q-Learning) é um algoritmo na qual a ideia é o aprendizado de como o agente deve se comportar e como deve mapear suas situações para suas ações, para melhorar as suas recompensas.

O agente inicia não sabendo que ações deve tomar mas necessita encontrar que ações vão encontrar melhores recompensas. Dessa forma, esse agente é passado por um treinamento interagindo com um ambiente e descobrindo os melhores caminhos para alcançar esse objetivo.

Diferente de escolher o melhor resultado visualizando as escolhas de curto prazo, o Q-Learning realiza escolhas levando em consideração as recompensas a longo prazo. O objetivo deste treinamento com Q-Learning é encontrar os passos de ações que maximizam a soma de reforços futuros, para o caminho mais curto do início do jogo até o objetivo final.

No trabalho, o agente foi o Pacman, o ambiente foi o labirinto, passou por 2000 treinamentos, seu objetivo foi terminar o jogo com a maior recompensa possível, sem ser “pego” pelo fantasma.

Esta fase é dividida em duas partes: **Treinamento**, na qual o pacman irá começar a aprender os valores de posições e as ações. Leva muito tempo para aprender os valores precisos, mesmo para esse labirinto pequeno, então o framework realiza essa fase em modo silencioso, sem apresentar o console. E **Teste**, na qual, desabilita o Q-learning e a exploração (turning off epsilon and alpha) e utilize sua política aprendida. Os jogos teste são mostrados no console ao final.

Arquivos alterados: qlearningAgents.py

Para trabalhar com o código-fonte alguns conceitos foram necessários para a equação do Q-Learning:

$$Q(\text{estado}, \text{ação}) = R(\text{estado}, \text{ação}) + \text{gama} * \text{Max}[Q(\text{próximo estado}, \text{todas ações})]$$

Onde,

- $Q(\text{estado}, \text{ação})$ é o valor da ação para um estado.
- gama é o fator de regularização, de desconto, possui um intervalo entre 0 e 1 para garantir a convergência da soma, podemos dizer que se estiver próximos a 0 tende a considerar recompensas imediatas, caso contrário irá considerar recompensas futuras com maior peso.
- α é o fator de aprendizagem, também possui um intervalo entre 0 e 1, na qual, próximo a 0 prioriza as informações anteriores e próximo a 1 prioriza informações recentes. está diretamente ligado a taxa de exploração.
- $R(\text{estado}, \text{ação})$ é a recompensa do estado por tomar uma ação.
- $\text{Max}[Q(\text{próximo estado}, \text{todas ações})]$ é a recompensa máxima por tomar a melhor ação.

No código alterado, `computeValueFromQValues` e `computeActionFromQValues` o algoritmo de Q-Learning utilizado permite que o agente verifique as quatro opções de direção e escolha a que tem a melhor recompensa e a ação com maior pontuação.

No método `getAction` a partir do retorno do método `util.flipCoin` submetendo a taxa de exploração podemos retornar duas ações para o agente, sendo uma um valor randômico das quatro ações possíveis (N,S,L,O) ou a ação recomendada provinda do `getPolicy`, que aplica a melhor política possível de escolha.

No método `update`, é atualizado próximo estado do agente levando em consideração que este estado exista no ambiente. Primeiramente é verificado se existe o próximo estado, caso não exista ele salva o aprendizado no estado atual, senão, calcula o aprendizado levando em consideração o desconto do fator de regularização e atualiza o estado atual.

Depois das alterações nos arquivos, realizamos os treinamentos. Observamos que o Pacman começou a aprender os primeiros valores das posições, ações e suas consequências. Foram realizados os testes abaixo, com os ajustes de parâmetros.

Parâmetros: **epsilon=0.05, gamma=0.8, alpha=0.2**

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

Telas do Treinamento	Treinamento do Pacman
----------------------	-----------------------



Figura 11: Treinamento do Pacman
Negativo

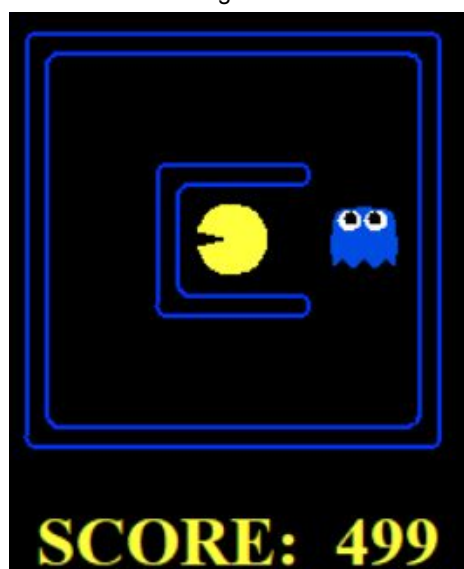


Figura 12: Treinamento do Pacman
Positivo

```
Beginning 2000 episodes of Training
Reinforcement Learning Status:
  Completed 100 out of 2000 training episodes
  Average Rewards over all training: -510.82
  Average Rewards for last 100 episodes: -510.82
  Episode took 0.57 seconds
Reinforcement Learning Status:
  Completed 200 out of 2000 training episodes
  Average Rewards over all training: -511.65
  Average Rewards for last 100 episodes: -512.49
  Episode took 0.79 seconds
....
Reinforcement Learning Status:
  Completed 1100 out of 2000 training episodes
  Average Rewards over all training: -287.29
  Average Rewards for last 100 episodes: -55.81
  Episode took 1.03 seconds
Reinforcement Learning Status:
  Completed 1200 out of 2000 training episodes
  Average Rewards over all training: -245.33
  Average Rewards for last 100 episodes: 216.20
  Episode took 1.08 seconds
....
Reinforcement Learning Status:
  Completed 1900 out of 2000 training episodes
  Average Rewards over all training: -80.42
  Average Rewards for last 100 episodes: 196.25
  Episode took 1.10 seconds
Reinforcement Learning Status:
  Completed 2000 out of 2000 training episodes
  Average Rewards over all training: -65.03
  Average Rewards for last 100 episodes: 227.39
  Episode took 1.05 seconds
Training Done (turning off epsilon and alpha)
-----
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 502
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 502
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Average Score: 498.8
Scores: 495.0, 495.0, 503.0, 502.0, 499.0, 503.0, 495.0,
502.0, 499.0, 495.0
Win Rate: 10/10 (1.00)
Record: Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

Rodando o programa descobrimos que os parâmetros da aprendizagem padrão que foram fornecidos pelo exercício foram eficazes. Jogamos um total de 2010 jogos e só os 10 últimos foram exibidos, nos quais mostra que o pacman aprendeu as políticas ideais para cada estado no ambiente.

Na lista dos resultados do treinamento podemos perceber na saída do desempenho do pacman que o seu desempenho só começa a melhorar depois dos 1100 primeiros jogos, a partir daí ele começa a ganhar mais do que perder, e ao final do treinamento podemos perceber que as recompensas do pacman são somente altas e positivas.

O gráfico 1 mostra as médias das pontuações ao longo da aprendizagem do agente Pacman.

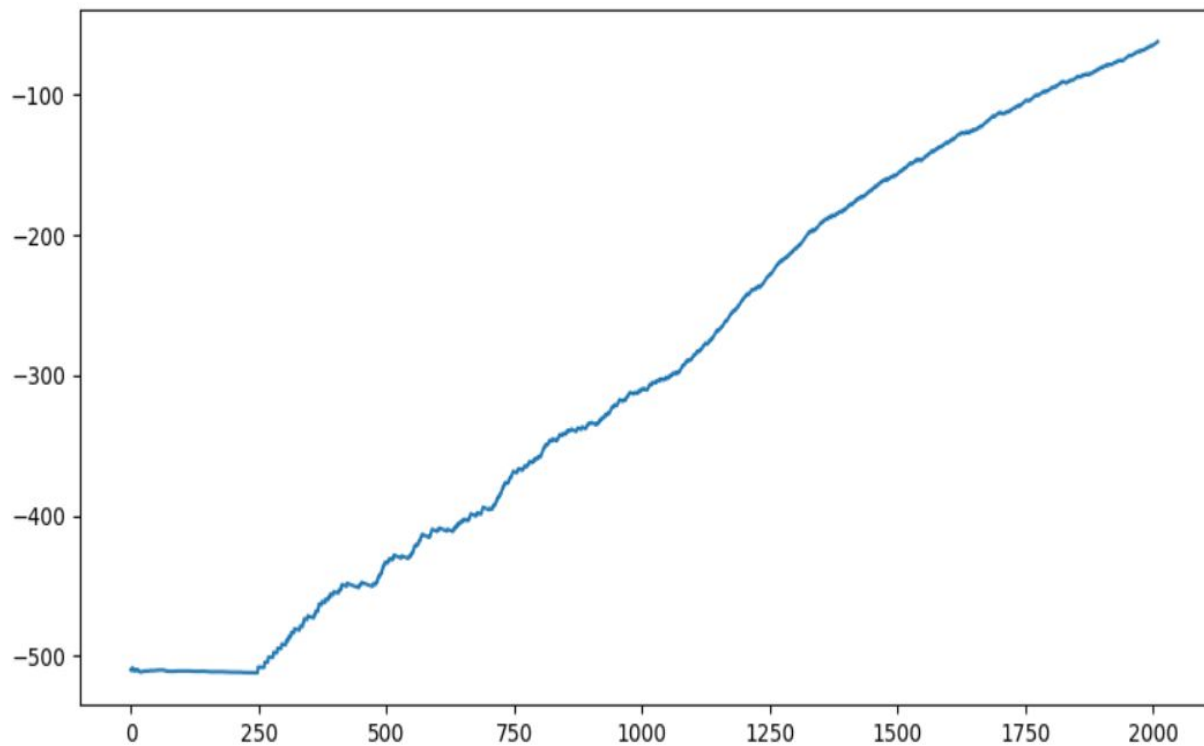


Gráfico 1: Média das pontuações ao longo da aprendizagem

Podemos observar no gráfico 1 que no início do treinamento a média da pontuação era baixa e à medida que os treinamentos foram acontecendo, o agente pacman obteve sucesso. Isso significa que à medida que o agente foi explorando o ambiente, o índice de recompensa foi gradualmente induzindo ele a evitar cometer erros, a aprendizagem média começou a melhorar a partir do treinamento 1100.

Testando outros parâmetros **epsilon=0.1,alpha=0.3,gamma=0.7**, neste treinamento pudemos perceber pelos resultados que também os parâmetros também foram eficazes, mas visualmente percebemos que o pacman explorou outros caminhos durante o treinamento, talvez foi por acaso, pois só visualizamos os 10 últimos treinamentos.

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

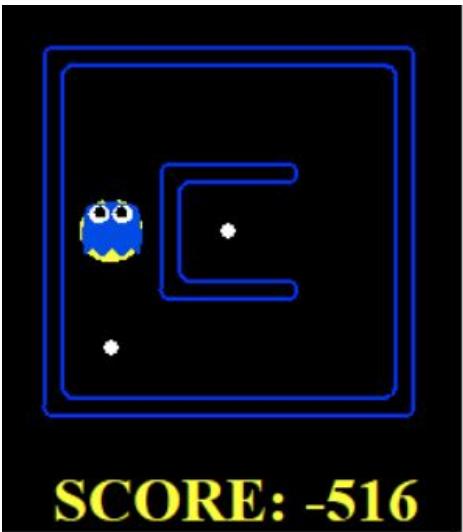

Telas do Treinamento	Treinamento do Pacman
	<p>Beginning 2000 episodes of Training</p> <p>Reinforcement Learning Status:</p> <ul style="list-style-type: none">Completed 100 out of 2000 training episodesAverage Rewards over all training: -512.26Average Rewards for last 100 episodes: -512.26Episode took 0.66 seconds <p>Reinforcement Learning Status:</p> <ul style="list-style-type: none">Completed 200 out of 2000 training episodesAverage Rewards over all training: -507.35Average Rewards for last 100 episodes: -502.43Episode took 0.82 seconds <p>....</p> <p>Reinforcement Learning Status:</p> <ul style="list-style-type: none">Completed 1000 out of 2000 training episodesAverage Rewards over all training: -300.25Average Rewards for last 100 episodes: -5.64Episode took 1.04 seconds <p>Reinforcement Learning Status:</p> <ul style="list-style-type: none">Completed 1100 out of 2000 training episodesAverage Rewards over all training: -268.86Average Rewards for last 100 episodes: 45.00Episode took 1.08 seconds <p>....</p> <p>Reinforcement Learning Status:</p> <ul style="list-style-type: none">Completed 1900 out of 2000 training episodesAverage Rewards over all training: -64.94Average Rewards for last 100 episodes: 257.66Episode took 1.06 seconds <p>Reinforcement Learning Status:</p> <ul style="list-style-type: none">Completed 2000 out of 2000 training episodesAverage Rewards over all training: -48.86Average Rewards for last 100 episodes: 256.62Episode took 1.08 seconds <p>Training Done (turning off epsilon and alpha)</p> <p>-----</p> <p>Pacman emerges victorious! Score: 503</p> <p>Pacman emerges victorious! Score: 503</p> <p>Pacman emerges victorious! Score: 499</p> <p>Pacman emerges victorious! Score: 499</p> <p>Pacman emerges victorious! Score: 503</p> <p>Pacman emerges victorious! Score: 503</p> <p>Pacman emerges victorious! Score: 503</p> <p>Pacman emerges victorious! Score: 495</p> <p>Pacman emerges victorious! Score: 495</p> <p>Pacman emerges victorious! Score: 495</p> <p>Average Score: 499.8</p> <p>Scores: 503.0, 503.0, 499.0, 499.0, 503.0, 503.0, 503.0, 495.0, 495.0, 495.0</p> <p>Win Rate: 10/10 (1.00)</p> <p>Record: Win, Win, Win, Win, Win, Win, Win, Win, Win, Win</p>
	

Figura 13: Treinamento do Pacman Negativo

Figura 14: Treinamento do Pacman Negativo

Ao final, segue um gráfico com as médias das pontuações ao longo da aprendizagem do agente Pacman.

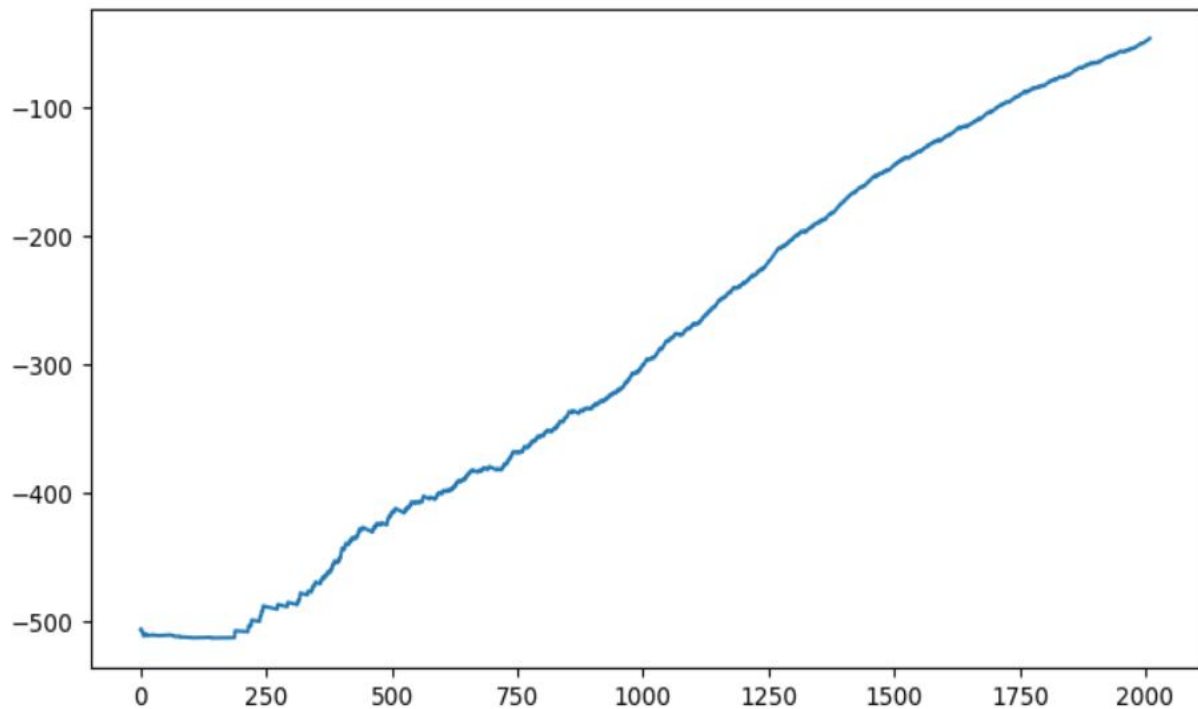


Gráfico 2: Média das pontuações ao longo da aprendizagem

Podemos observar no gráfico 2 que, apesar de aumentar a taxa de exploração e de aprendizado, não houve aumento significativo na média das pontuações com relação ao gráfico 1, a aprendizagem média começou a melhorar a partir do treinamento 1000.

Outros treinamento do pacman com os parâmetros iniciais de $\epsilon=0.05$, $\gamma=0.8$ e $\alpha=0.2$, no entanto ele perdeu todas as vezes, o que mostra é que na hora do teste o pacman não explorou todo o ambiente do jogo, percebemos também que o treinamento demorou muito mais que o anterior. Então o pacman não consegue vencer em um labirinto maior pois cada labirinto requer uma configuração diferente, por mais que se aumente a taxa de exploração e se altere o fator de aprendizado valorizando as recompensas imediatas, o agente foca em explorar um ambiente pequeno.

```
python pacman.py -p PacmanQAgent -x 5000 -n 5010 -l mediumGrid
```



Figura 15: Treinamento do Pacman Negativo

Como podemos observar no gráfico 5, da média das pontuações ao longo da aprendizagem para o labirinto mediumGrid, o agente não consegue aprender no labirinto maior, mantendo uma média de -508 depois de 2000 treinamentos.

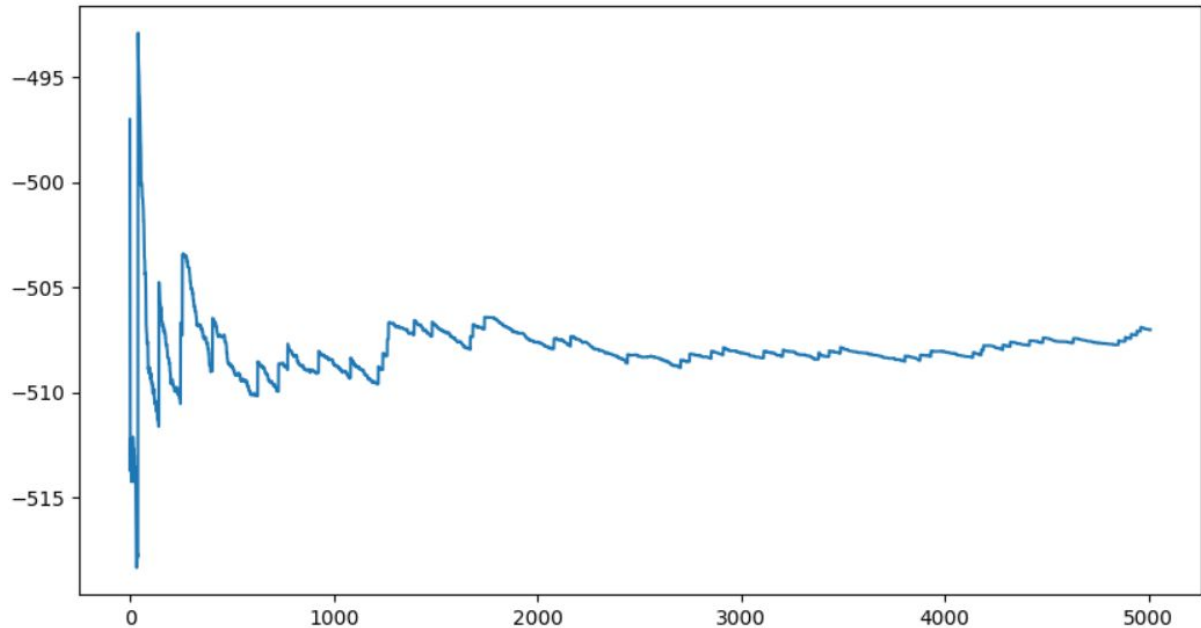


Gráfico 5: Média das pontuações ao longo da aprendizagem para o mediumGrid

Avaliando a questão

Ao final da questão o exercício sugere que teste a questão com o seguinte código para verificar a eficiência dos algoritmos

```
python autograder.py -q q9
```

```
Reinforcement Learning Status:
  Completed 100 test episodes
  Average Rewards over testing: 500.60
  Average Rewards for last 100 episodes: 500.60
  Episode took 1.31 seconds
Average Score: 500.6
Scores:      495.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 499.0, 499.0, 503.0, 503.0,
495.0, 503.0, 503.0, 503.0, 499.0, 503.0, 495.0, 499.0, 499.0, 503.0, 495.0, 499.0, 503.0, 503.0,
503.0, 503.0, 499.0, 495.0, 499.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 495.0, 495.0,
499.0, 499.0, 503.0, 503.0, 503.0, 495.0, 499.0, 503.0, 503.0, 503.0, 495.0, 503.0, 499.0, 495.0,
503.0, 499.0, 495.0, 503.0, 503.0, 495.0, 499.0, 499.0, 503.0, 503.0, 503.0, 495.0, 503.0, 503.0,
503.0, 499.0, 499.0, 495.0, 503.0, 499.0, 499.0, 503.0, 503.0, 495.0, 499.0, 503.0, 503.0, 503.0,
499.0, 499.0, 503.0, 495.0, 503.0, 503.0, 503.0, 499.0, 503.0, 503.0, 495.0, 503.0, 499.0, 503.0,
503.0, 503.0, 503.0, 495.0
Win Rate:    100/100 (1.00)
Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win
*** PASS: test_cases\q9\grade-agent.test (1 of 1 points)
*** Grading agent using command: python pacman.py -p PacmanQAgent -x 2000 -n 2100 -l smallGrid -q
-f --fixRandomSeed
```

```
***      100 wins (1 of 1 points)
***      Grading scheme:
***      < 70:  0 points
***      >= 70:  1 points
```

Considerações Finais

O objetivo deste trabalho foi implementar, experimentar e utilizar o algoritmo de aprendizado por reforço (Reinforcement Learning) para melhorar o comportamento do agente Pacman no ambiente (labirinto), utilizando o framework da berkeley.edu como base para o experimento.

Após a implementação, testes e calibragem dos parâmetros para o programa funcionar.

O primeiro experimento mostrou que com a taxa de aprendizado e de exploração padrão, o agente Pacman aprende as políticas ideais para cada estado do ambiente e seu desempenho melhora a partir dos 1100 primeiros jogos.

O segundo experimento mostrou que com a taxa de aprendizado e de exploração ambas aumentadas, o agente Pacman aprende as políticas ideais para cada estado do ambiente e seu desempenho melhora a partir dos 1000 primeiros jogos.

Também realizamos testes em outros labirintos, porém percebemos que o treinamento demorou muito mais que o anterior. Então o pacman não consegue vencer em um labirinto maior pois cada labirinto requer uma configuração diferente.

Um estudo posterior poderia avaliar o desempenho do agente nos mapas maiores, incluindo o mapa original do jogo, com essas calibragem específicas.

Um ponto fraco é o fato dos estados estarem representados de forma fixa, poderia ser sugerido que se apresentassem as fases de forma randômica. Claro que isso também pode causar certas limitações, pois o método falha em generalizar os cenários que não encontrou durante o treinamento. Mas seria interessante testar o treinamento nesses mapas novos aleatórios.

Apesar dessas limitações, este trabalho provou a eficiência do algoritmo de aprendizado por reforço (reinforcement learning - Q-Learning), e esse método pode ser aplicado a muitos outros problemas.