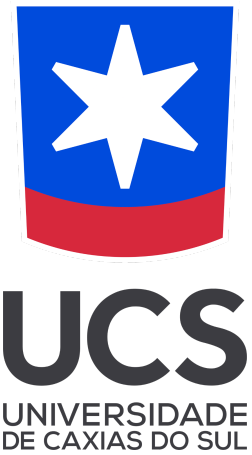


Universidade de Caxias do Sul
Nomes: Douglas Biazus
Professor(a): Carine Geltrudes Webber

Projeto Avaliativo
Aprendizado por Reforço:

Navegação em Labirintos com Q-Learning

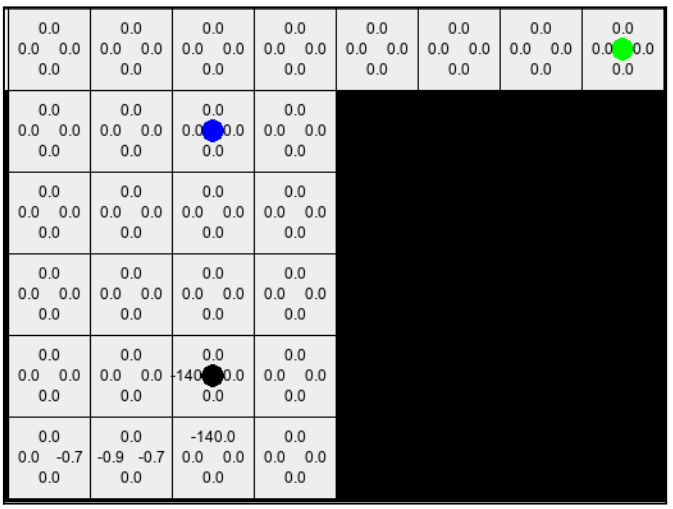
Caxias do Sul
2025

Introdução

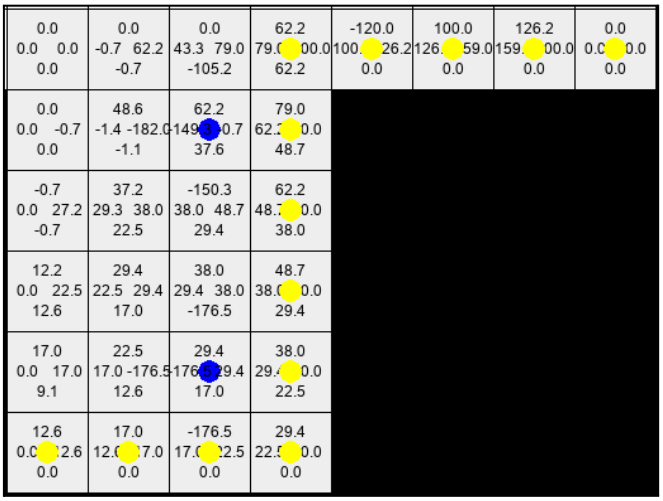
Como trabalho final de semestre, foi proposto implementar o algoritmo Q-Learning, com interface gráfica, para que um agente aprenda a navegar em um labirinto usando o algoritmo. Para o desenvolvimento do trabalho, foi utilizado a linguagem de programação Python, juntamente com a biblioteca Pygame para criação da visualização com interface gráfica. O programa executa múltiplas simulações para que o agente aprenda a encontrar o melhor caminho, recebendo pontuações positivas ao chegar no objetivo, mas perdendo pontuação ao passar por obstáculos, incentivando a encontrar o caminho mais eficiente.

Q-Learning

Dos algoritmos de aprendizado por reforço, há os que aprendem com o que é realizado pelo agente (*on-policy*) e os que aprendem como o que deveria ter sido feito (off-policy). O Q-learning é um algoritmo off-policy. Mesmo realizando uma ação não ótima, a tabela Q é atualizada com base na melhor ação para aquele estado, para que em próximas iterações escolhas melhores sejam feitas, buscando um melhor caminho no caso da navegação em labirintos.



(*Foto de parte da visualização em uma das primeiras iterações)*
(Note que há pontuação negativa pelo agente ter se deslocado para cima, movimento não ótimo.)



(*Foto de parte da visualização na iteração final.) (2ª casa decimal ocultada)*
(O agente prioriza a direção com maior valor na tabela Q.)

O algoritmo possui uma tabela em que os valores são atualizados conforme a fórmula:

*Q[estado][ação] = Q[estado][ação] + alfa * (recompensa + gama * max(Q[próximo_estado]) - Q[estado][ação])*

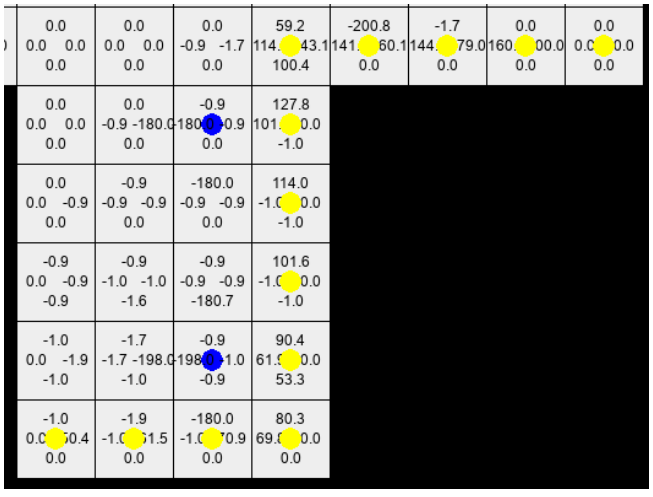
• Alfa é a taxa de aprendizado, que determina o peso das novas iterações na Tabela Q. Um Alfa alto acelera o aprendizado do agente, fazendo ele dar mais peso a novas informações, porém podendo comprometer a qualidade da Tabela Q, enquanto que um Alfa baixo prioriza estabilidade, obrigando mais iterações e tornando o processo lento.

• Gama é o fator de desconto, com base no próximo estado (quadrado) que o agente estará presente. Um Gama alto faz o agente priorizar a recompensa futura. Um Gama baixo, faz priorizar a recompensa na posição atual.

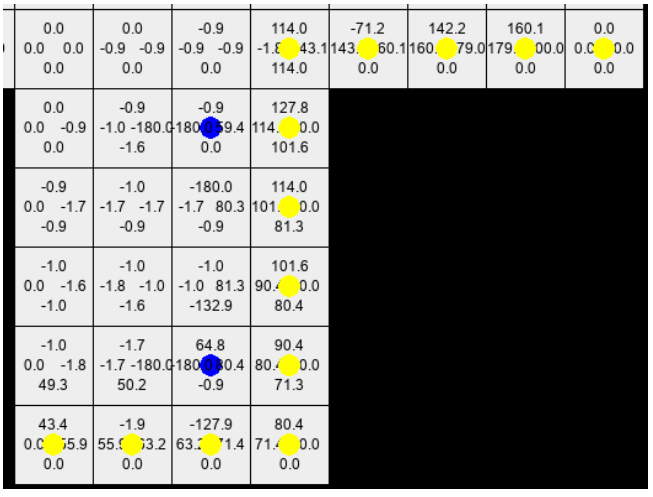
Por fins de simplicidade, alfa e gama fixos foram utilizados no projeto. Testei o algoritmo diminuindo alfa em cada etapa, e usando alfa e gama mais altos, mas o resultado não sofreu alterações. O agente consegue alcançar o objetivo em menos de 3000 iterações independente dos valores testados, exceto ao diminuir o gama, porque um gama baixo reduz a importância da recompensa futura. No caso do labirinto atrapalhou na priorização do caminho mais curto.

Métricas

Em termos de métricas, por volta de 50 iterações, o agente já apresenta um bom desempenho, errando o caminho apenas pela aleatoriedade de 30% imposta. Essa aleatoriedade de 30% é necessária pois sem ela o agente poderia ficar preso em caminhos “vai-e-volta” nas primeiras iterações, não aprendendo. Foi necessário um mínimo de 18 iterações para que a Tabela Q mostrasse um caminho até o objetivo.



(*Foto de parte da visualização na 18ª iteração.)*



(*Foto de parte da visualização na 50ª iteração.)*

Aplicações Práticas de Q-Learning

Por armazenar cada par de estado e ação em uma linha diferente, ou matriz diferente, o Q-Learning não é eficiente para aplicações com muitos dados. No jogo Five-in-a-Row, Gomoku, “cinco em linha” que tentei desenvolver, o número de estados possíveis e combinações no tabuleiro é gigante, obrigando a diminuir o número de casos. Só é possível de se fazer isso fazendo o Q-Learning de outras maneiras, como tratando linhas de “cuidado” e “ameaça de fim de jogo” ao invés de tratar o tabuleiro, limitando o Q-Learning e obrigando a criar mais funções com funcionamento do jogo para lidar com a recompensa. Não é possível criar uma punição e deixar o algoritmo aprender por conta o funcionamento do jogo igual é no labirinto. Se fossemos aplicar para um sistema de recomendação de filmes, também teríamos o mesmo problema: não dá para usar uma combinação de filmes assistidos e filmes novos para fazer recomendação com Q-Learning pelo volume de informações. Precisaríamos segmentar o problema em vários menores, fazendo um algoritmo para descobrir categorias de filmes (comédia, drama, ação, suspense) que o usuário tem interesse, outro para descobrir quais diretores de filmes recomendar filmes, e outro, por fim, para mostrar quais filmes as pessoas mais gostam em geral, e não individualmente. Simplesmente não é viável.

O Q-Learning é bom para resolver problemas simples, que dá pra dividir tudo em passos definidos, e em poucas ações. O jogo Snake em que a cobra precisa evitar bater nas paredes e buscar comida, e o jogo Pac-Man, que o agente precisa aprender a fugir dos fantasmas, ganhar pontos, pegar frutas, e terminar a fase o mais rápido, são dois exemplos de jogos que imagino serem boas aplicações do QLearning. Em controles de robôs, robótica? Talvez para evitar obstáculos, em mapas fixos (semelhante ao labirinto do projeto, um aspirador de pó evitando bater em objetos da casa) Em controle autônomo? Talvez para saber quais luzes ligar e desligar, levando em conta o horário e os hábitos dos moradores, para simular que há pessoas na casa mesmo quando a família está de viagem, aumentando a segurança. Fora desses cenários simples, o Q-Learning se torna muito limitado e não consegue lidar com ambientes complexos.