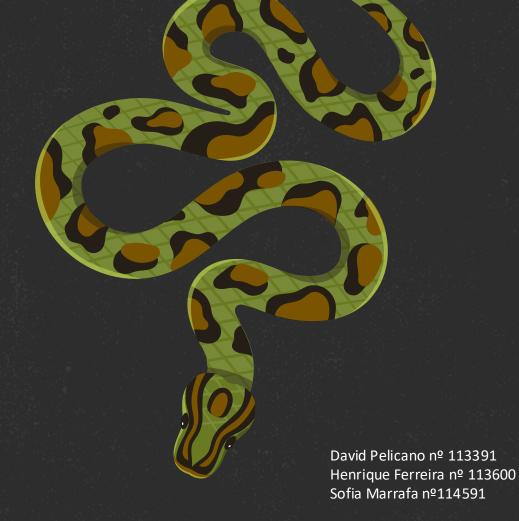


Jogo da cobra Python





Processo

- 1. Marca posições visíveis como visitadas
- 2. Identifica as posições não visitadas e atualiza a lista marked_places
- 3. Calcula, para cada coordenada não visitada, o número de posições já visitadas e adiciona-as à lista **I_points**.
- 4. As quatro posições adjacentes à cabeça da cobra são armazenadas na lista closest_coords.
- 5. Caso seja detetada comida, a posição é adicionada à lista **food_position**.

Resultados

- O **I_points** é ordenada de forma crescente, priorizando as menos visitadas
- O closests_coords é ordenada com base no número de coordenadas
- food_position é ordenada pela distância até a cabeça da cobra, sendo a primeira posição a mais próxima.



Decisão de Movimento

A partir das listas geradas, o algoritmo segue uma sequência de verificações até determinar a próxima meta (food).

Prioridades na escolha da próxima meta

- 1. Comida detetada
 - Se existir, food=food_position[0]
- 2. Exploração de posições não descobertas
 - Caso contrário, food=l_points[0]
- 3. Processo para posições antigas
 - Na ausência de posições descobertas, food=marked_places[0]

Gestão de posições visitadas

Limite de 700 posições visitadas Após atingir o limite, posições mais antigas são removidas







Geração de pontos aleatórios

De forma a garantir que a cobra não fique presa ou sem opções viáveis de movimento

- São gerados 3 pontos aleatórios fora do campo de visão atual da cobra
- Estes pontos servem como possíveis extensões do caminho até ao destino
- A escolha do ponto aleatório final depende das verificações e buscas realizadas

Cenários de decisão do caminho

- Caminho Completo: Encontrar um caminho da cabeça até o objetivo (food) e, em seguida, do objetivo até um dos pontos aleatórios.
- Caminho Parcial: Encontrar o caminho até o objetivo, mas falhar ao conectar o objetivo a um dos pontos aleatórios.
- Sem caminho: Não encontrar nenhum caminho até o objetivo.

Próxima coordenada

- Caminho Completo/Parcial: coord = solution_path[len(body)]
- Sem caminho : coord = closest_coords[0]







- Algoritmo Escolhido: A cobra utiliza o A* que encontra o caminho até ao destino(s) e calcula a distância de Manhattan entre os pontos para escolher o caminho mais eficiente
- O **Procura Eficiente:** O algoritmo considera o custo de cada caminho, ajustando a procura para evitar pontos de alto custo

Evitar Colisões

- O Durante a procura, a cobra evita colisões com o próprio corpo e com outras cobras
- O algoritmo leva em conta o movimento da cobra, permitindo que ela passe por posições que inicialmente faziam parte do corpo, mas que posteriormente deixam de ser

Eficiência da Procura

- O **Consideração de custo:** O algoritmo A* ajusta a procura com base no custo de cada caminho
- O **Busca de menor custo:** Caso o custo exceda 1000, a busca é interrompida e considerada inviável, otimizando o desempenho do programa



Conclusão

O agente desenvolvido navega efetivamente no ambiente do jogo, demonstrando a capacidade de tomar decisões estratégicas de forma autónoma. Ao aproveitar algoritmos de busca e processar eficientemente os estados do jogo, o agente maximiza a sua pontuação enquanto se adapta a condições dinâmicas. Futuras melhorias podem envolver a otimização do processo de busca, incorporação de aprendizagem automática para movimentos preditivos e lidar com cenários de jogo mais complexos com múltiplos agentes.

