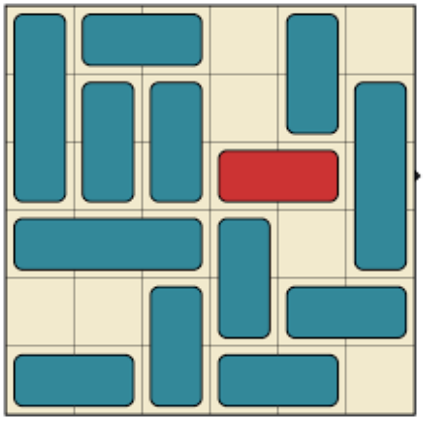
Inteligência Artificial

universidade de aveiro



Rush-Hour

Projeto Realizado por:
-> Rafael Santos : 98466
-> Guilherme Craveiro :103574

1

Estrutura da solução

agente.py	solver.py	veiculo.py	common.py
Script que inicia a comunicação com o servidor, um objeto do tipo Solver e executa a função *MainLoop*	Script que contém as principais classes desenvolvidas. Classe Solver inclui a lógica da solução, implementada com recurso ao BFS. Classe Mapa, derivada da classe Map para implementar funcionalidades específicas.	Script que contém a classe Veiculo. Classe Veiculo mantém os dados de cada veículo do *puzzle*.	Script fornecido do qual foram utilizadas as classes: Coordinates e Map.

2

Classe Veiculo

- Esta classe tem os seguintes dados do veículo:
  - id (letra que identifica o veículo);
  - coords (as coordenadas da posição do primeiro elemento do veículo);
  - movimento (tipo de movimento que o veículo pode fazer);
  - tamanho (número de casas que o veículo ocupa).

- Implementa a função __str__ que devolve todos os dados em formato de string.

3

Classe Mapa

Classe derivada da classe Map que implementa as seguintes funcionalidades:

- O construtor da classe faz uma lista de todos os veículos existentes;
- Uma função que devolve as coordenadas de um qualquer veículo;
- Validação de coordenadas;
- Verificação dos "vizinhos" de um veículo;
- Uma função *iterator* que é responsável por devolver todos os movimentos possíveis dos veículos;
- Esta classe é utilizada para os diferentes estados do puzzle mas também para converter um estado em movimentos dos veículos, comparando um estado com o seguinte é possível saber qual o veículo que se moveu e em que direção.

4

Classe Solver

- Mantém a comunicação com o servidor, envia os movimentos e recebe o mapa atualizado do servidor.
- Ao receber o mapa do servidor:
  - Caso não exista nenhuma solução:
    - Gera soluções para o puzzle
    - Escolhe a melhor solução (que é a que tem menos passos)
  - Caso exista uma solução:
    - Compara o mapa recebido do servidor com a versão local:
      - Se for diferente:
        - Procura o puzzle recebido nas soluções e passos existentes para o caso de já existir um passo igual e assim não ser necessário gerar novas soluções
        - Se não existir gera soluções novas
      - Se não for diferente passa para o próximo passo da solução
- Envia os comandos para o servidor com base nos passos da solução escolhida

5

Algoritmo BFS

- O algoritmo escolhido para a solução foi o Breadth-First Search;
- Implementado na classe Solver na função breadth_first_search;
- A função pode utilizar objetos ou tuplos (parâmetro optimize);
- A solução inicial com objetos revelou-se muito lenta, só conseguindo resolver os primeiros 5/6 puzzles;
- A solução final utiliza tuplos.

6

Algoritmo BFS

- O algoritmo utiliza a função NextMove para gerar os diferentes estados da solução;
- Cada estado é uma cópia do puzzle com um veículo movido;
- Os movimentos são adicionados à fila dos movimentos a testar;
- Cada movimento é depois removido da fila e adicionado a um caminho com os passos que possivelmente resolvem o puzzle;
- Cada caminho é testado e adiciona à lista de caminhos visitados para evitar repetições desnecessárias;
- Se o passo adicionado ao caminho posiciona o carro do jogador na posição de terminar o puzzle os passos da solução são guardados como sendo uma solução válida;

7

Algoritmo BFS

- O algoritmo implementado pode ser interrompido antes de encontrar uma solução em função de dois parâmetros:
  - max_depth - determina a profundidade máxima da solução a procurar;
  - max_time - define o tempo, em segundos, que no máximo pode ser despendido na procura da solução
- Quando não encontra uma solução o algoritmo guarda os passos que estava a explorar;
- O algoritmo pode gerar várias soluções sendo escolhida a que tiver menos passos – parâmetro nr_solucoes e função MelhorSolucao;

8

Algoritmo BFS

- O algoritmo termina quando:
  - Testou todos os movimentos possíveis;
  - Encontrou o número de soluções indicadas no parâmetro nr_solucoes;
  - Atingiu o limite de profundidade da pesquisa definido no parâmetro max_depth;
  - Ultrapassou o tempo máximo atribuído ao parâmetro max_time;
- Quando a solução é encontrada é guardada para que cada passo da solução seja convertido em movimentos de veículos. (função ConverterSolucaoEmMovimentos)
- Para cada movimento primeiro é necessário mover o cursor até ao veículo pretendido e só depois é que é executado. (função MoveCursorToCarAndSelect)
- Para testar o programa foi necessário acrescentar um mecanismo de *logging* para ficheiro por forma a verificar os erros e o tempo despendido em cada passo na procura da solução.

9

Bibliografia

GitHub - ryanwilsonperkin/rushhour: A heuristic-based solver for the Rush Hour game - https://github.com/ryanwilsonperkin/rushhour

The Complexity of Sliding-Block Puzzles and Plank Puzzle, Robert A. Hearn

A First Course on Data Structures in Python, Donald R. Sheehy

10