Inteligência Artificial

Trabalho Prático de Grupo – Rush Hour

Prof. Diogo Gomes
Prof. Luís Seabra Lopes

Artur Correia (102477)
Daniel Carvalho (77036)

Estratégia Utilizada

Na implementação desenvolvida foram aplicados os algoritmos Breadth-First (grelhas <=6x6) e Greedy (grelhas >6x6)

O algoritmo de pesquisa foi implementado na classe SearchTree, em **tree_search.py**, tendo sido baseado na árvore utilizada nas aulas práticas. No entanto, foram aplicadas as seguintes alterações:

- A definição de Problema, Domínio e Nó foi feita com recurso a tuplos, em vez de classes.
- Os nós abertos foram guardados nas listas open_nodes e all_nodes, com a primeira a guardar o ID e a segunda o próprio nó, num índice equivalente.
- Na expansão dos novos estados correspondentes às ações possíveis sobre um nó, apenas são considerados estados que não tenham sido anteriormente visitados, sendo estes guardados no set visited.

Domínio

- As funções utilizadas na definição do domínio foram implementadas em **domain.py**.
- Nesta implementação, considerou-se que o Estado do problema pode ser descrito por um tuplo que guarda a representação da grelha em string, e o seu tamanho. Deste modo, o funcionamento das funções descritas é baseado em operações sobre strings, o que torna o modelo bastante eficiente em relação à estratégia adotada na entrega anterior
- Em **func_actions**, são determinadas as ações a aplicar para um dado estado. São consideradas como ações os movimentos de uma unidade possíveis para cada carro, (para casas desocupadas).
- Em **func_result**, é determinado o estado resultante da ocorrência de uma ação, sendo para tal, na posição do carro movido após a ação, substituído o caracter correspondente na string.
- Em **func_satisfies**, é verificado se o estado atual da grelha corresponde ao objetivo, sendo para tal verificado se a posição do carro A se encontra no final da sua linha.

Heurística

A estratégia de heurística, implementada em **func_heuristic**, considera dois fatores para o cálculo do custo heurístico de um estado:

- O cálculo da distância do carro A ao final da sua linha, sendo para tal considerado apenas o tamanho da grelha e o índice de A na string do Estado (H_1)
- A determinação do número de células vazias na linha do carro A, realizando operações sobre a string que define o Estado (H_2)

$$H = H_1 - H_2$$

- O valor de H_2 é subtraído ao de H_1 , uma vez que a existência de células vazias na linha de A fornece mais deslocações possíveis a este. Isto permite mais possibilidades de movimentação de outros carros, abrindo o caminho para A atingir o final da linha
- Foi considerado apenas o custo heurístico, e não o custo das ações (para implementação de A* em vez de Greedy), uma vez que se verificou que o tempo de execução nesse caso era sempre ligeiramente superior

Student.py

- A execução da pesquisa em árvore retorna as jogadas necessárias para completar o nível, sendo para cada determinado o movimento do carro correspondente (moves)
- Em cada iteração de **agent_loop**, é determinado o próximo movimento do cursor de forma a completar a jogada atual (cabeça da lista moves)
- O facto de cada iteração do loop corresponder a apenas 1 movimento do cursor permite que o Agente raramente fique dessincronizado do servidor, sendo que quando tal acontece ele consegue facilmente voltar a um estado que permite a resolução do nível, sem necessidade de recalcular a árvore
- A exceção verifica-se na ocorrência de crazy cars. Isto é detetado quando se observa que a grelha recebida do servidor é diferente do que era esperado após a realização da jogada anterior, sendo então ativada a estratégia de replaneamento
- A estratégia de replaneamento consiste em recalcular a árvore, uma vez que foi verificado que isto é mais eficiente do que outras alternativas (como por exemplo, voltar a colocar o carro movido na posição inicial)

Benchmark

- Para a realização dos testes de benchmark da solução, foi aplicado o script **test_tree.py**, que corre o algoritmo de pesquisa para um dado mapa.
- Foram testados os níveis de maior pontuação para as diferentes possibilidades de tamanho de grelha. Para cada teste comparou-se a eficiência dos algoritmos Breadth-First e Greedy.

Tempos de Execução, em s, dos diferentes algoritmos de pesquisa.

	Nível 11 (8x8)	Nível 38 (6x6)
Breadth-First	14.892354696999973	1.1460867919997781
Greedy	0.01027895299897228	4.400760039000033

- Foi então decidida a aplicação do algoritmo Greedy em grelhas >6x6, e Breadth-First noutros casos.
- O highscore obtido na execução do **student.py** nos nossos computadores pessoais foi 1557439.