Relatório de SI

Realizado por:
Grupo 01
Bernardo Melo 50025
Saul Silva 44301

Espaco de estados

Para a definição do nosso Ambiente, definimos uma classe Estado. Esta classe tem um construtor que recebe como parametros:

def init__(self,width,height,local_arrumador,local_alvos,local_caixas,local_paredes)

onde:

. local_arrumador uma instância de Arrumador((x,y))

.local_caixas uma lista de instâncias de Caixa((x,y))

.local_alvos uma lista de instâncias de Alvo((x,y))

.local_paredes uma lista de instâncias de Parede((x,y))

Ainda nesta classe definidos os seguintes métodos:

.percept(self,thing): que dado uma thing devolve um dicionário com todoas as things á volta, a uma distância de um quadrado. Exemplo: {"cima":Caixa((x,y)),"baixo":Alvo((x,y))}

.in_corner(self,thing): que devolve true se uma thing está num canto e false cc.

__eq__(self) para podermos testar a igualdade de estados

__lt__(self) para podermos comparar estados, usado no algoritmo de heurísticas

__hash__(self)

Definicao do problema

Para a definição do problema criamos uma classe Game que estende de Problem, Esta classe tem um construtor que recebe como parâmetros:

def __init__(self,inicial):

onde:

.inicial é o estado inicial

Tivemos ainda de definir os seguintes métodos:

actions(self,state) método abstrato da classe Problem que dado um estado devolve uma lista com as possíveis acções .

result(self,state,action) método abstrato da classe Problem que dado um estado e uma acção devolve o estado resultante de aplicar ao estado a acção.

Observações

Algoritmos em árvore: Os algoritmos em árvore não encontram solução pois não fazem a distinção dos estados, ou seja, o algoritmo não compara estados, logo não sabe se já visitou um certo estado ou não. Na prática o arrumador fica “encalhado” num cima/baixo eterno.

Mapa 1

```
#####
#o..#
#.*.#
#..A#
#####
```

Algoritmos	**Nós expandidos**	**Tempo de execução**	**Nº Jogadas**	**Solução**
depth_first_tree_search	N/A	N/A	N/A	Não tem solução possível.
depth_first_graph_search	127	0.072	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
breadth_first_tree_search	N/A	N/A	N/A	Não tem solução possível.
breadth_first_search	53	0.025	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
uniform_cost_search	50	0.029	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
Depth_limited_search	70	0.03	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
iterative_deepening_search	166	0.079	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
Greedy_best_first_graph_search - H1	23	0.011	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']

Greedy_best_first_graph_search - H2	102	0.051	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
Greedy_best_first_graph_search - H3	42	0.021	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
Greedy_best_first_graph_search - H4	25	0.024	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
Astar_search - H1	23	0.013	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
Astar_search - H2	77	0.039	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
Astar_search - H3	34	0.017	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']
Astar_search - H4	23	0.016	5	['mover_cima', 'empurra_esq', 'mover_baixo', 'mover_esq', 'empurra_cima']

- **Depht_limited_search:** neste algoritmo percebemos que o melhor nível de profundidade,i.e, na prática, número de passos, é 5 pois é o menor número de passos que o arrumador tem de fazer para encontrar a solução.
- **Iterative_deepening_search**: Neste algoritmo percebemos que é o tem maior expansão de nós e também o menos eficiente para o "puzzle1.txt".
- **Depth_first_graph_search**: Este algoritimo é o segundo menos eficiente porque é o que necessita de expandir mais nós até conseguir chegar a um resultado. No entanto, a sua rapidez em encontrar a solução é relativamente boa.
- **Breadth_first_search**: Neste algoritmo de largura percebemos que ele consegue terminar em relação ao *"Breadth_first_tree_search" porque ele não volta a ver os estados anterior. Enquanto o outro algoritmo pode entrar em ciclo infinito..*
- **Uniform_cost_search**: Neste algoritmo ele procura o caminho com o custo menor até ao estado final. Este algoritmo poderia ser mais eficiente se a função *"path_cost" fosse melhorada.*
- **Greedy_best_first_graph_search**:Neste algoritmo sabemos que olha para o nó que tiver a menor heurística. Deste modo, é possível perceber que a nossa heurística H2 é menos eficiente que as restantes.
- **A***: Este algoritmo para além de olhar para as heurísticas olha para o custo do caminho e como é possível perceber é o algoritmo mais eficiente.

Para o Mapa 1 o algoritmo mais eficiente é o A* com as heurísticas H1 e H4. Não é possível dizer qual o melhor dado que ambos expandem os mesmos nós e com o tempo de aproximação muito próximo.

Mapa 2

```
#######
##....#
##*.o.#
#..A..#
#.*#o.#
#..####
#######
```

Algoritmos	Tempo de execução	Nós expandidos	Nº Jogadas	Solucao
depth_first_tree_search	N/A	N/A	N/A	Não tem solução possível.
depth_first_graph_search	7.502	10167	131	['mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'mover_direita', 'mover_cima', 'mover_cima', 'mover_esq', 'mover_esq', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'empurra_direita', 'mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_direita', 'mover_cima', 'empurra_esq', 'empurra_esq', 'mover_cima', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'empurra_direita', 'mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_direita', 'mover_cima', 'empurra_esq', 'mover_baixo', 'empurra_esq', 'mover_direita', 'mover_direita', 'mover_cima', 'mover_cima', 'mover_esq', 'mover_esq', 'mover_esq', 'mover_baixo', 'empurra_direita', 'mover_esq', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_direita', 'mover_cima', 'mover_cima', 'mover_esq', 'mover_esq', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'empurra_direita', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'empurra_baixo', 'mover_direita', 'mover_cima', 'mover_cima', 'mover_esq', 'mover_esq', 'mover_esq', 'mover_baixo', 'empurra_direita']
breadth_first_tree_sear	N/A	N/A		Não tem solução possível.

ch				
breadth_first_search	7074	5.534	43	['mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'empurra_baixo', 'mover_esq', 'mover_esq', 'mover_cima', 'empurra_direita']
Uniform_cost_search	7361	6.386	43	Igual ao breadth_first_search
Depth_limited_search	N/A	N/A	N/A	Não encontrou solução com depth limit de 50.
iterative_deepening_search	N/A	N/A	N/A	N/A
Greedy_best_first_graph_search - H1	1219	0.978	43	['mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'empurra_baixo', 'mover_esq', 'mover_esq', 'mover_cima', 'empurra_direita']
Greedy_best_first_graph_search - H2	7649	8.053	51	['mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_cima', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_direita', 'mover_cima', 'empurra_esq', 'mover_cima', 'mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'empurra_direita', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_baixo', 'mover_esq', 'mover_esq', 'mover_cima', 'empurra_direita']
Greedy_best_first_graph_search - H3	3106	4.168	45	['mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_cima', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'empurra_esq', 'mover_cima', 'mover_esq', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_direita', 'mover_cima', 'mover_cima', 'mover_esq', 'mover_esq', 'empurra_baixo', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo',

				'empurra_esq', 'mover_cima', 'mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'empurra_direita', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_baixo', 'mover_esq', 'mover_esq', 'mover_cima', 'empurra_direita']
Greedy_best_first_graph_search	564	0.436	43	['mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'empurra_baixo', 'mover_esq', 'mover_esq', 'mover_cima', 'empurra_direita']
Astar_search - H1	962	0.753	43	['mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'empurra_baixo', 'mover_esq', 'mover_esq', 'mover_cima', 'empurra_direita']
Astar_search - H2	7260	8.270	43	
Astar_search - H3	3566	4.994	43	
Astar_search - H4	556	0.451	43	['mover_esq', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'mover_baixo', 'mover_esq', 'empurra_cima', 'mover_esq', 'mover_cima', 'mover_cima', 'mover_esq', 'empurra_baixo', 'empurra_baixo', 'mover_esq', 'mover_baixo', 'mover_baixo', 'mover_direita', 'empurra_cima', 'mover_esq', 'mover_cima', 'empurra_direita', 'empurra_direita', 'mover_cima', 'mover_cima', 'mover_direita', 'mover_direita', 'mover_baixo', 'empurra_esq', 'empurra_baixo', 'mover_esq', 'mover_esq', 'mover_cima', 'empurra_direita']

Algorítmos:

- **Depth_first_graph_search** : Encontra a solução mas demora algum tempo e a solução não é de longe a melhor. Isto pode-ser dar por o algoritmo percorrer em profundidade todas as opções possíveis sem repetir estados anteriores, mas em certas ocasiões seria mais eficiente se o algoritmo repetisse estados anteriores.
- **Depht_limited_search**: Experimentamos com vários límites de profundidade e reparámos que quanto menor for o limite menos estados ele vai explorar, se não encontrar o estado final até ao limite de profundidade para a execução. Na nossa opinião achamos que o algoritmo eventualmente chegará à solução se definirmos um limite de profundidade adequado.
- **Uniform_cost_search:**
- **Iterative_deepening_search**: Ao contrário do *deph_limited_search* este algoritmo vai incrementando mais um no limite de profundidade até encontrar o resultado. Tal como no *deph_limited_search* achamos que é preciso chegar a um nível muito grande de profundidade para chegar à solução.
- **Greedy_best_first_graph_search:** este algoritmo para expandir um nó olha para o nó que tiver um menor valor heurístico na fronteira. Este algoritmo é bastante eficiente e rápido

- **A*:** este algoritmo para além de olhar as heurísticas referidas em baixo também olha para o custo do caminho que percorreu até ao nó.

Para o Mapa 2 o algoritmo mais eficiente é o A* com a heurísticas H4.

Heurísticas

H1: esta heurística devolve a distância real(i.e,em linha reta) de cada caixa ao alvo mais perto. Bastante eficiente.
H2: esta heurística devolve o número de caixas que estão nos cantos. Portanto valoriza os estados que não têm nenhuma caixa no canto. Esta não é uma heurística muito boa para ser usada sozinha pois todos os outros estados que estão na fronteira, que não têm caixas no canto, vão ser escolhidos “aleatoriamente” (por ordem da lista da actions).
H3: esta heurística devolve a distância real do arrumador à caixa mais perto de si.
H4: esta heurística é a junção da H1 e H2. Foi a melhor heurística que encontramos pois combina a h1 que é a melhor heurística única para casos gerais com a h2 que é boa para casos específicos.

Nota1: casos gerais, i.e, para todos os estados
Nota2: casos específicos, i.e, para estados específicos

Testámos todas as combinações entre heurísticas, nomeadamente a junção de todas, mas nenhuma das combinações foi melhor do que a h4.

Algoritmos de procura cega vs Algoritmos de heurísticas

No geral os algoritmos de heurísticas vencem em rapidez, desempenho e eficiência aos algoritmos de procura cega para além de que alguns algoritmos de procura cega não encontram a solução(profundidade_arvore,largura_arvore), isto porque nos algoritmos de heurísticas nós definimos a ordem pela qual é escolhido um nó na fronteira com base no que é melhor para ele resolver o jogo.