UNIVERSIDADE
DE ÉVORA

# Relatório do 1º trabalho
# Inteligência Artificial

**Trabalho realizado por:**

**- Henrique Rosa, nº 51923**

**- Diogo Matos, nº 54466**

**24/03/2024**

# 1 - Inicialização

Todos os estados do problema podem ser representados pelo seguinte predicado: e(X,Y,P,Q,N), no qual intuitivamente:

- Y representa o número da linha onde se encontra o Agente;
- X representa o número da coluna onde se encontra o Agente;
- Q representa o número da linha onde se encontra a Máquina;
- P representa o número da coluna onde se encontra a Máquina:
- N representa a lista de pares de posição (X,Y) dos objetos

Os obstáculos serão definidos pelo predicado x consoante a sua posição:

x( X,Y ). representaria um obstáculo (x) na posição (X,Y)

**a)** Podemos então representar cada um dos exemplos em prolog com o devido estado inicial e final das seguintes formas, listadas na tabela abaixo:

| 1º Exemplo | 2º Exemplo | 3º Exemplo |
|---|---|---|
| **estado_inicial(e(2,1,2,2,[(2,4)])).**<br>**estado_final(e(_,_,2,7,[ ])).**<br><br>**x( 4,2 ).**<br>**x( 4,3 ).**<br>**x( 4,4 ).**<br>**x( 1,6 ).**<br>**x( 3,6 ).**<br>**x( 7,6 ).**<br>**x( 4,7 ).** | **estado_inicial(e(2,1,2,2,[(5,5)])).**<br>**estado_final(e(_,_,5,7,[ ])).**<br><br>**x( 4,2 ).**<br>**x( 4,3 ).**<br>**x( 3,4 ).**<br>**x( 1,6 ).**<br>**x( 3,6 ).**<br>**x( 3,7 ).**<br>**x( 7,6 ).** | **estado_inicial(e(2,1,2,2,[(2,4),(6,2)])).**<br>**estado_final(e(_,_,5,7,[ ])).**<br><br>**x( 4,2 ).**<br>**x( 4,3 ).**<br>**x( 4,4 ).**<br>**x( 3,7 ).**<br>**x( 3,6 ).**<br>**x( 1,6 ).**<br>**x( 7,3 ).**<br>**x( 7,6 ).** |

**b)** O agente pode mover-se nas direções cima, baixo, esquerda e direita. Se a máquina estiver na direção do movimento do agente, o agente empurra-a na mesma direção do seu movimento. Caso a caixa esteja numa posição onde se encontra um objeto **e o agente estiver adjacente a ela**, a máquina pode apanhar o objeto. O agente não pode entrar nas posições com obstáculo (x) mas é também de notar que a máquina também não pode, pois por definição, se a máquina entrar numa casa com obstáculo e o agente a tentar empurrar, fica o agente na casa com obstáculo.

Para cumprir cada uma das ações listadas, foi definido no ficheiro op.pl o predicado **op/3** e os predicados auxiliares **iguais/2** e **bounds/2** com as seguintes definições:

```
op(e(X,Y,P,Q,N),baixo, e(X, Y1, P, Q1,N), 1) :-
   Y1 is Y - 1,
   (iguais(e(X, Y1), e(P, Q)) ->
      (
         Q1 is Q - 1,
         bounds(P, Y1),
         \+ x(P, Q1)
      );
      (
         Q1 is Q,
         bounds(X, Y1),
         \+ x(X, Y1)
      )
   ).
```

```
op(e(X,Y,P,Q,N),direita,e(X1,Y,P1,Q,N),1) :-
   X1 is X+1,
   (iguais(e(X1,Y),e(P,Q)) ->
      (
         P1 is P+1,
         bounds(P1,Q), bounds(X1,Y),
         \+ x(P1,Q)
      );
      (
         P1 is P,
         bounds(X1,Y),
         \+ x(X1,Y)
      )
   ).
```

```
op(e(X,Y,P,Q,N),cima,e(X,Y1,P,Q1,N),1) :-
   Y1 is Y+1,
   (iguais(e(X,Y1),e(P,Q)) ->
      (
         Q1 is Q+1,
         bounds(P,Q1), bounds(X,Y1),
         \+ x(P,Q1)
      );
      (
         Q1 is Q,
         bounds(X,Y1),
         \+ x(X,Y1)
      )
   ).
```

```
op(e(X,Y,P,Q,N),esquerda,e(X1,Y,P1,Q,N),1) :-
   X1 is X-1,
   (iguais(e(X1,Y),e(P,Q)) ->
      (
         P1 is P-1,
         bounds(P1,Q),  bounds(X1,Y),
         \+ x(P1,Q)
      );
      (
         P1 is P,
         bounds(X1,Y),
         \+ x(X1,Y)
      )
   ).
```

```
op(e(X,Y,P,Q,N),apanha,e(X,Y,P,Q,N1),1) :-
      (abs(X-P) + abs(Y-Q))<2,
      member((P,Q),N),
      delete(N,(P,Q),N1).
```

```
bounds(X, Y) :-
   X =< 7,
   X >= 1,
   Y =< 7,
   Y >= 1.

iguais(A, A).
```

**c)** O código para a resolução desta alínea encontra-se no ficheiro "pni.pl", sendo apenas o código de pesquisa não informada dado nas aulas com algumas modificações para a representação no terminal ficar mais agradável.

Tiremos como exemplo o exemplo 1 do enunciado para resolver este problema. a solução está na profundidade 6 (assumindo que a raiz está na profundidade 0), pois são necessárias 6 ações para chegar à solução (5 movimentos para cima mais apanhar o objeto). sabendo que cada nó pode despoletar 5 outros nós diferentes (ir para cima, baixo, esquerda, direita e apanhar objeto caso possível), e assumindo o pior dos casos em que o nó da solução está o mais à direita possível na árvore de pesquisa, podemos então estimar o número de nós visitados e em memória de cada um dos seguintes algoritmos:

- **Pesquisa em profundidade:** Sabendo que a pesquisa neste problema pode dar origem a ciclos (por exemplo o agente mexer-se da casa (1,1) para a casa (1,2) e vice versa infinitamente), o número de nós visitados e em memória, no pior dos casos, seria infinito, tornando este algoritmo inviável para este tipo de problema

- **Pesquisa em profundidade iterativa:** $5^6$ nós visitados,
  $5^6$ em memória.

- **Pesquisa em largura:** $5^6$ nós visitados,
  $5^7 - 5$ nós em memória.

Como vamos notar mais á frente, um dos maiores problemas com que nos deparamos foi a falta de memória para resolver o problema, resultando as execuções em stack overflow, pelo que temos um viés para a quantidade de nós em memória, e então escolhemos o algoritmo de pesquisa em profundidade iterativa como algoritmo mais eficiente.

**d)** Na tabela abaixo encontram-se os valores que obtivemos para a resolução de cada um dos exemplos com o Algoritmo de pesquisa em profundidade iterativa (de notar que o código que utilizamos para a pesquisa faz a contagem de nós visitados em todas as iterações cumulativamente).

| | **Estados Visitados** | **Estados em Memória** | **Profundidade** | **Custo** |
|---|---|---|---|---|
| **Exemplo 1** | 36306 | 15625 | 6 | 6 |
| **Exemplo 2** | Stack Overflow | - | - | - |
| **Exemplo 3** | Stack Overflow | - | - | - |

Os Exemplos 2 e 3 deram stack overflow, pelo que foi-nos impossível obter a contagem de nós visitados e em memória.

**e)** As heurísticas que propomos são a distância de Manhattan e a distância euclidiana, ambas a contar a distância desde a máquina até à posição de saída. Estas duas heurísticas são admissíveis porque, caso não houvesse objetos, o custo estimado seria igual ao custo real para a máquina simplesmente chegar ao destino, e no caso de haver objetos, as heurísticas não os contam, fazendo assim o caminho real mais longo do que a estimativa, o que faz com que estas duas medidas sejam majoradas pelo custo real em qualquer ponto do problema, tornando-as assim admissíveis. O código destas duas heuristicas está presente no ficxheiro op.pl , sendo estes o seguinte:

```
manhattan(e(_,_,P,Q,_),C):-
        estado_final(e(_,_,P1,Q1,_)),
        A is abs(P - P1),
        B is abs(Q - Q1),
        C is A+B.
```

```
euclidiana(e(_,_,P,Q,_),SOMA):-
        estado_final(e(_,_,P1,Q1,_)),
        SOMA is round(sqrt(abs(P -
        P1)** 2 + abs(Q - Q1) ** 2)).
```

**f)** O código para a resolução desta alínea encontra-se no ficheiro “pi.pl”, sendo apenas o código de pesquisa informada dado nas aulas com algumas modificações para a representação no terminal ficar mais agradável.

O algoritmo que achamos mais eficiente para este problema é o algoritmo A*, pois, ao contrário do greedy, este faz a estimativa do custo para cada nó com o custo real até o nó presente, sendo teoricamente mais eficiente na pesquisa pela solução.

Já para a melhor heurística, escolhemos a distância de manhattan, pois esta apresenta uma estimativa de custo muito mais realista comparativamente a distância euclidiana, que iria calcular a distância “real”, o que não é útil neste problema porque o movimento dos elementos do problema seguem uma grelha.

**g)**Na tabela abaixo encontram-se os valores que obtivemos para a resolução de cada um dos exemplos com o Algoritmo A* com a heurística de Manhattan:

| | **Estados Visitados** | **Estados em Memória** | **Profundidade** | **Custo** |
|---|---|---|---|---|
| **Exemplo 1** | 25 | 67 | 6 | 6 |
| **Exemplo 2** | 286 | 467 | 13 | 13 |
| **Exemplo 3** | Stack Overflow | - | - | - |

O exemplo 3 deu stack overflow, pelo que foi-nos impossível obter a contagem de nós visitados e em memória.