

## Relatório do 1º Projeto

### Grupo 5

#### Problema:

- O problema descrito pelo projeto envolve um jogo de tabuleiro com uma grelha quadrada. Neste tabuleiro, existem robots com diferentes cores e um alvo com uma certa cor. O objetivo é deslocar até ao alvo o robot que tem a mesma cor, tendo em conta que não podem estar vários robots na mesma posição e que existem barreiras de modo a proibir a passagem. Os robots apenas podem fazer movimentos retos em 4 direções (para cima, para baixo, para a direita ou para a esquerda).

#### Solução:

- A nossa solução baseia-se na utilização de dicionários, uma vez que é uma estrutura de dados fácil e rápida de aceder através das suas *keys*.

- Utilizamos o método de procura *iterative deepening search*, uma vez que, após vários testes, foi o método que nos fez obter melhor resultados em relação ao tempo despendido.

#### Resultados obtidos:

##### Procura em largura primeiro:

→ Uma vez que este método gera todas as sequências possíveis, a memória utilizada é exponencial, apesar de ser um método completo.

##### Procura em profundidade primeiro:

→ Utilizando este método, entramos em *loop* infinito (não é uma procura completa), uma vez que os nós visitados continuam a aparecer sempre que se expande a árvore. Assim, expande-se sempre um nó que já foi visitado previamente, nunca terminando. Concluindo, este método não é completo.

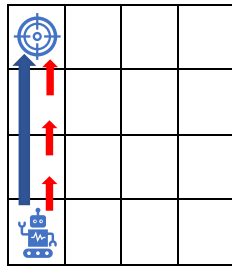
##### Procura gananciosa:



→ Este método faz a melhor escolha a cada *step*, porém, muitas das vezes não produz a solução ótima. Ao testar várias instâncias, verificámos que esta procura pode gerar mais movimentos do que as soluções ótimas.

##### A\*:

→ Ao implementarmos a nossa heurística, esta providenciou-nos melhores resultados em testes “pequenos”, porém, como não era admissível, demorava muito tempo a terminar testes maiores.

Esta heurística baseava-se em calcular a distância do robot até ao alvo e um exemplo simples desta heurística devolver um valor superior ao custo real é:



Seja  o alvo e  o robot;

Não existindo barreiras nem robots entre eles, a heurística devolve que o custo é 3, porque o robot tem de se deslocar 3 quadrículas para cima até chegar ao alvo (setas a vermelho). Porém, no contexto do nosso projeto, o robot apenas tem de fazer um movimento (ação = (B, u)) (seta a azul).

Concluindo, a heurística é maior que o custo real em alguns casos e assim, a heurística não é admissível e o método A\* não é ótimo.

#### Procura em profundidade iterativa:

- Esta procura foi a que nos providenciou melhores resultados a nível de memória e de tempo. Apesar desta procura expandir e gerar muitos mais nós do que o método A\*, para testes maiores concretizou ótimos tempos e a memória ocupada foi razoável. Esta procura é completa.

#### Avaliação experimental dos resultados:

Gerámos vários testes aleatórios e como exemplo, vamos mostrar os resultados obtidos para 2 deles.

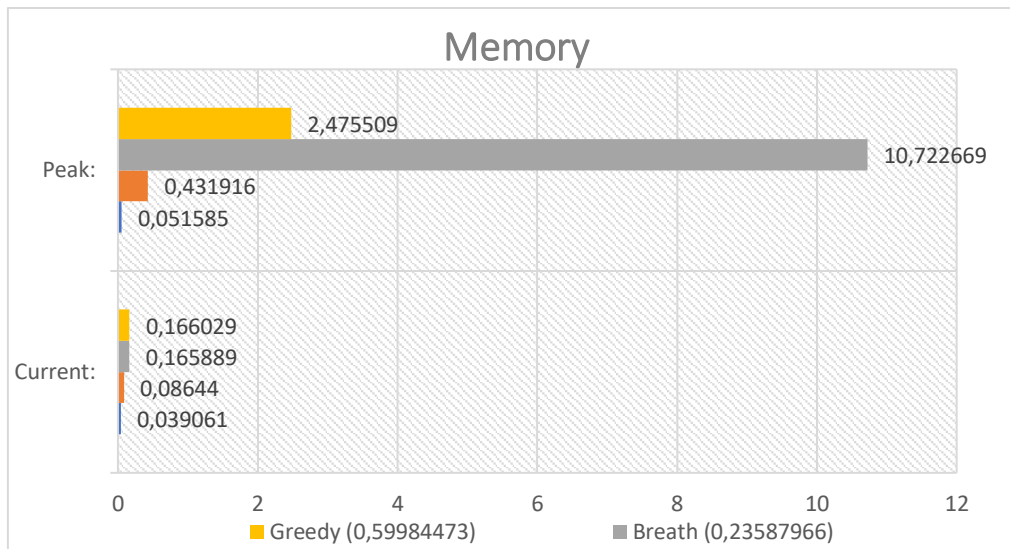
No teste público i1.txt, obtivemos melhores resultados para o método A\* (uma vez que é um teste pequeno, como explicámos previamente).

Os resultados obtidos para os métodos A\*, *Breadth-First Search*, *Greedy search* e *Iterative Deepening search* encontram-se no anexo 1 (memória) e no anexo 2 (nós expandidos, nós testados, nós gerados e tempo).

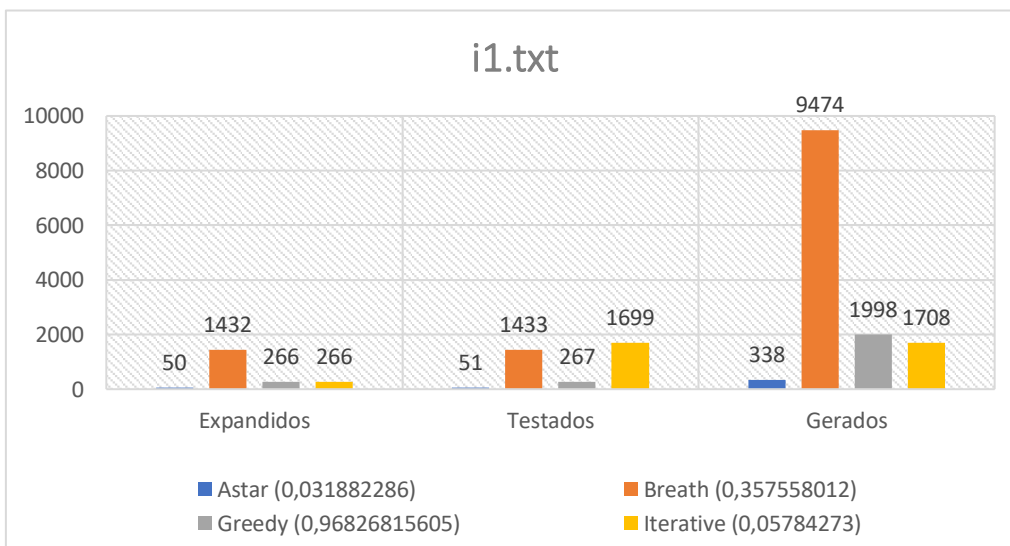
Para um teste gerado por nós com maior dimensão e um maior número de barreiras, o método *greedy* não terminou num tempo razoável e deste modo, apenas apresentamos os valores para os métodos A\*, *Breadth-First Search* e *Iterative Deepening search*. Como explicámos previamente, a heurística aplicada na procura A\* não é admissível e assim, os resultados para este teste são piores do que para os outros métodos, como era previsto por nós. Os resultados encontram-se no anexo 3.

Após uma análise cuidada do nosso código e dos métodos de procura utilizados, concluímos que os resultados obtidos eram os esperados por nós.

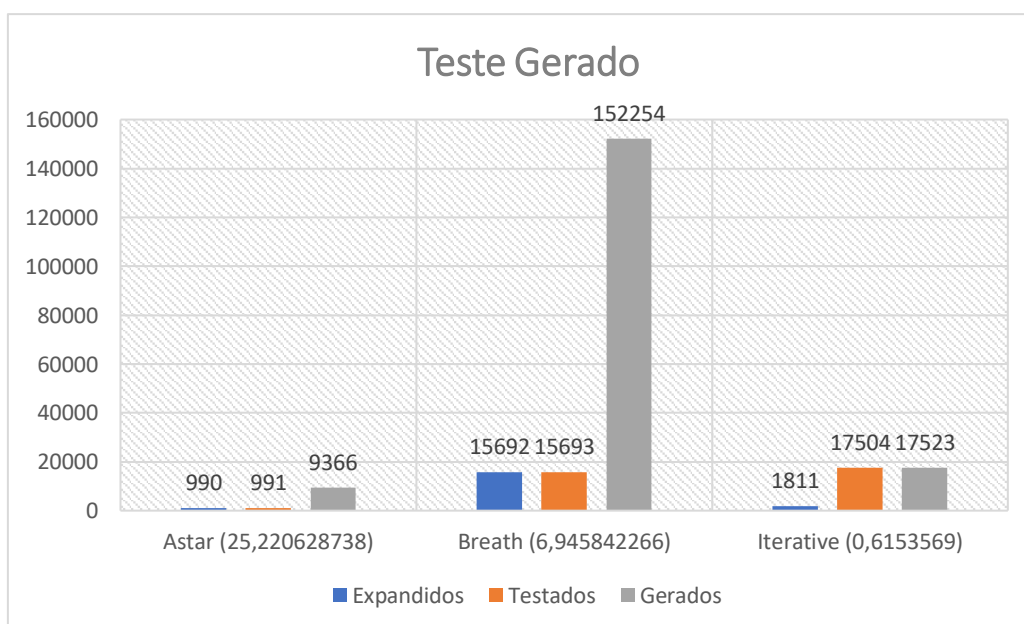
**Nota:** Os resultados do método *Depth-First Search* não se encontram nos nossos gráficos, uma vez que entrou em *loop* infinito para todos os testes.



Anexo 1



Anexo 2



Anexo 3