Grupo Nº 21



**Inteligência Artificial**

1.º Semestre 2015/2016

**IA-Tetris**

Relatório de Projecto

Índice

[1 Implementação Tipo Tabuleiro e Funções do problema de Procura 3](#_Toc437358135)

[1.1 Tipo Abstracto de Informação Tabuleiro 3](#_Toc437358136)

[1.2 Implementação de funções do problema de procura 3](#_Toc437358137)

[2 Implementação Algoritmos de Procura 3](#_Toc437358138)

[2.1 Procura-pp 3](#_Toc437358139)

[2.2 Procura-A\* 3](#_Toc437358140)

[2.3 Outros algoritmos 3](#_Toc437358141)

[3 Funções Heurísticas 3](#_Toc437358142)

[4 Estudo Comparativo 4](#_Toc437358143)

[4.1 Estudo Algoritmos de Procura 4](#_Toc437358144)

[4.1.1 Critérios a analisar 4](#_Toc437358145)

[4.1.2 Testes Efectuados 4](#_Toc437358146)

[4.1.3 Resultados Obtidos 4](#_Toc437358147)

[4.1.4 Comparação dos Resultados Obtidos 5](#_Toc437358148)

[4.2 Estudo funções de custo/heurísticas 5](#_Toc437358149)

[4.3 Escolha da procura-best 5](#_Toc437358150)

# Implementação Tipo Tabuleiro e Funções do problema de Procura

## Tipo Abstracto de Informação Tabuleiro

Para representar o tipo tabuleiro optámos por usar um array bidimensional. A escolha desta estrutura de dados advém da simplicidade na execução de operações, de criação e modificação em comparação a outras alternativas, como por exemplo listas, onde estas operações não seriam tão triviais.

## Implementação de funções do problema de procura

A função accoes é responsável por devolver uma lista com todas as accoes possíveis que podem ser feitas com a próxima peça a ser colocada. Para tal, verifica todas as rotações válidas e todas as posições onde estas podem ser colocadas.

A função resultado vai retornar um novo estado que resulta de aplicar a accao recebida ao estado original. Primeiro verifica em que linha vai colocar a peça da accao, criando um estado resultante da aplicação da peça no tabuleiro do estado inicial. De seguida verifica se o tabuleiro esta preenchido até ao topo, se não estiver verifica se existem linhas totalmente preenchidas e remove-as do tabuleiro, atualizando a pontuação de acordo com a quantidade de linhas removidas.

# Implementação Algoritmos de Procura

## Procura-pp

A nossa procura em profundidade primeiro foi implementada de forma recursiva. Para isso começa por verificar se o nó atual é solução do problema, se for devolve as accoes necessárias ate chegar ao nó solução; se não for, vai expandir o nó atual e aplica uma accao ao estado atual criando assim outro estado. Repete enquanto não encontrar solução, caso não exista retorna uma lista vazia.

## Procura-A\*

Na procura A\* é guardado na fronteira não só os estados a ainda serem explorados como também as accoes desde o estado inicial ate ao estado atual e o valor da função de avaliação associado. Cada vez que que é guardado um valor na fronteira é necessário reorganizar toda a fronteira. Como a fronteira está sempre organizada o próximo nó a ser explorado vai ser sempre o primeiro da fronteira.

## Outros algoritmos

Não implementámos outros algoritmos de procura.

# Funções Heurísticas

Uma vez que demos prioridade em primeiro lugar em passar aos testes do Mooshak, a implementação das heurísticas foi guardada para a parte final da elaboração do projeto. No entanto, não foi possível gerirmos de forma eficiente o nosso tempo e concentrar esforços na construção de heurísticas. Apesar disso, foi possível passar a 58 testes de 61 no Mooshak, uma vez que na procura- best usámos uma procura A\*com a adição de uma heurística simples, que apenas retorna 0, acabando por não acrescentar nada à procura A\*.

# Estudo Comparativo

## Estudo Algoritmos de Procura

### Critérios a analisar

Nesta comparação decidimos usar o tempo, a pontuação obtida e o espaço ocupado como critérios de comparação, visto que são os fatores mais importantes neste projeto. O tempo e o espaço são fatores limitantes e temos como objetivo maximizar a pontuação obtida.

### Testes Efectuados

Para este estudo, usámos um teste fornecido pelo corpo docente (Teste 19 E2) modificado por nós, mantendo o tabuleiro parcialmente preenchido e usando a procura-pp e a procura-A\*.

### Resultados Obtidos

Tabela 1 – Valores obtidos da aplicação do teste fornecido

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Procura-pp | | | Procura-A\* | | |
| Nº peças | Tempo (s) | Pontuação | Espaço (mb) | Tempo (s) | Pontuação | Espaço (mb) |
| 3 | 0.001 | 0 | 0.029 | 0.006 | 700 | 0.334 |
| 6 | 0.001 | 0 | 0.064 | 0.009 | 1100 | 0.549 |
| 9 | 0.001 | 100 | 0.096 | 0.156 | 1300 | 5.644 |
| 12 | 0.015 | 100 | 0.155 | 1.718 | 1400 | 218.294 |
| 15 | 0.078 | 100 | 4.761 | 11.116 | 1500 | 1732.812 |

Gráfico 2 – Tempo em funcão do número de peças usando o algoritmo Procura A\*

Gráfico 1 – Tempo em funcão do número de peças usando o algoritmo Procura-pp

Gráfico 4 – Espaço em funcão do número de peças usando o algoritmo Procura-A\*

Gráfico 3 – Espaço em funcão do número de peças usando o algoritmo Procura-pp

### Comparação dos Resultados Obtidos

Como é possível observar pelos valores obtidos a procura-pp é bastante mais rápida que a procura-A\*, no entanto visto que a procura-pp só se preocupa em encontrar uma solução ela não terá valores ótimos em relação à pontuação. Em contraste, a procura-A\* tenta otimizar localmente a sua pontuação, sendo por isso muito mais demorada, mas em contrapartida obtém valores muito melhores em termos de pontuação. Como a procura-A\* guarda todos os nós visitados, abertos e a lista de ações até eles, precisará de muito mais espaço em comparação com a procura-pp que apenas guarda o caminho até ao nó atual.

## Estudo funções de custo/heurísticas

Como não implementámos nenhuma heurística não temos valores para fazer este estudo.

## Escolha da procura-best

Na procura-best usámos a procura – A\* com uma heurística. Devido à falta de tempo a heurística que implementámos retorna sempre 0, não fazendo nada a mais que a procura A\*.