# Introdução

O principal objetivo deste projeto visa a modelação de uma variação do jogo *Same Game* como um **problema de satisfação**. O objetivo deste jogo é remover conjuntos de bolas adjacentes da mesma cor, de maneira a chegar a um tabuleiro vazio.

A obtenção de soluções para variados tabuleiros de *Same Game* foi efetuada aplicando três algoritmos de procura diferentes: Procura em Profundidade Primeiro (DFS), Procura Gananciosa e o algoritmo A\*.

Para os algoritmos de procura informada, foi necessário estabelecer uma heurística que acelerasse a escolha de nós a expandir e a obtenção de uma solução. Foi escolhida como heurística o número de grupos removíveis em cada estado, por se tratar de uma estimativa fiável do número de ações necessárias para atingir uma solução. Em casos de empate, é usado o número total de peças restantes no tabuleiro.

# Resultados

Foram corridos vários testes para cada algoritmo (cada tabuleiro foi corrido 5 vezes para os algoritmos Ganancioso e A\*), exceto para a Procura em Profundidade Primeiro (DFS) (apenas uma corrida por tabuleiro). Os tabuleiros testados são os 6 apresentados na secção 3 do enunciado, os tabuleiros 1 e 3 não têm solução:

**Tabuleiro 1** – 4x5, 2 cores **sem solução**: [[1, 2, 1, 2, 1], [2, 1, 2, 1, 2], [1, 2, 1, 2, 1], [2, 1, 2, 1, 2]]

**Tabuleiro 2** – 4x5 com 3 cores: [[1, 2, 2, 3, 3], [2, 2, 2, 1, 3], [1, 2, 2, 2, 2], [1, 1, 1, 1, 1]]

**Tabuleiro 3** – 10x4, 3 cores **sem solução**: [3, 1, 3, 2], [1, 1, 1, 3], [1, 3, 2, 1], [1, 1, 3, 3], [3, 3, 1, 2], [2, 2, 2, 2], [3, 1, 2, 3], [2, 3, 2, 3], [5, 1, 1, 3], [4, 5, 1, 2]]

**Tabuleiro 4** – 10x4, 3 cores: [[3, 1, 3, 2], [1, 1, 1, 3], [1, 3, 2, 1], [1, 1, 3, 3], [3, 3, 1, 2], [2, 2, 2, 2], [3, 1, 2, 3], [2, 3, 2, 3], [2, 1, 1, 3], [2, 3, 1, 2]]

**Tabuleiro 5** – 10x4, 5 cores: [[1, 1, 5, 3], [5, 3, 5, 3], [1, 2, 5, 4], [5, 2, 1, 4], [5, 3, 5, 1], [5, 3, 4, 4], [5, 5, 2, 5], [1, 1, 3, 1], [1, 2, 1, 3], [3, 3, 5, 5]]

Os resultados apresentam-se no quadro abaixo. A quantidade de nós gerados indicada não inclui o nó inicial.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algoritmo** | **Tabuleiro** | **succs** | **goal\_tests** | **states** | **tempo (seg)** |
| DFS | 1 | 1 | 1 | 0 | 0.00000 |
| 2 | 4 | 5 | 7 | 0.01563 |
| 3 | 74702 | 74702 | 74701 | 48.92422 |
| 4 | 54 | 55 | 85 | 0.04688 |
| 5 | 3123308 | 3123339 | 3123363 | 1620.63681 |
| Greedy | 1 | 1 | 2 | 0 | 0.00000 |
| 2 | 3 | 5 | 6 | 0.01563 |
| 3 | 74702 | 74703 | 74701 | 57.56526 |
| 4 | 42 | 44 | 59 | 0.04688 |
| 5 | 6293 | 6295 | 6356 | 3.95332 |
| A\* | 1 | 1 | 2 | 0 | 0.00000 |
| 2 | 3 | 5 | 6 | 0.00000 |
| 3 | 74702 | 74703 | 74701 | 47.68979 |
| 4 | 42 | 44 | 59 | 0.04688 |
| 5 | 5116 | 5118 | 5278 | 3.51580 |

# Análise dos Resultados

* Dos 3 algoritmos de procura testados, aquele que obteve melhores resultados, em geral, foi o A\*. Isto deve-se principalmente ao uso de uma heurística fiável que reduz significativamente o número de nós expandidos.
* No pior caso, a complexidade temporal tanto da DFS como do A\* é de O(bd), em que *b* é o número médio de sucessores por estado e *d* é a profundidade do caminho que conduz à solução. A superioridade do A\* em relação à DFS é evidente no número de nós expandidos e gerados, que é muito maior no caso da DFS.
* A procura gananciosa produziu resultados satisfatórios, semelhantes aos produzidos pelo A\*, pois para este problema não existe solução ótima (i. e., não é possível encontrar o menor número de ações possível que conduza a uma solução). Portanto, a existência de uma heurística fiável é o necessário para encontrar uma solução eficientemente, já que a solução encontrada pela procura gananciosa pode não ser a ótima. A complexidade temporal da procura gananciosa é também idêntica à da DFS e do A\*: O(bd).
* A complexidade da execução de cada algoritmo aumenta significativamente ao aumentar o tamanho do tabuleiro. Este aumento é exponenciado quando o número total de cores aumenta, uma vez que resulta num maior número de grupos removíveis (e, portanto, nós expandidos).
* A complexidade temporal da rotina board\_find\_groups é O(N), em que N é o tamanho do tabuleiro. Esta rotina é semelhante a uma *Depth-First Search*: percorre recursivamente todo o tabuleiro, visitando as posições adjacentes e adicionando-as aos respetivos grupos, com o cuidado de verificar se uma posição já foi visitada.
* A complexidade temporal da rotina board\_remove\_group é O(N), em que N é o tamanho do tabuleiro. Esta rotina percorre todas as posições do grupo dado, removendo a peça lá colocada e realinhando vertical e horizontalmente a disposição das restantes peças. No pior caso, um grupo corresponde a um tabuleiro inteiro.