UNIVERSIDADE EDUARDO MONDLANE
FACULDADE DE CIÊNCIAS
CURSO DE INFORMÁTICA

Inteligência Artificial

8 Puzzle – A*

Discentes:
Simão Gabriel Mazive
Inercio Bélton Fanequiço
Frank Salvador Cumaio

Docente:
Orlando Zacarias

Abril, 2017

UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE CIÊNCIAS

CURSO DE INFORMÁTICA

Inteligência Artificial

Avaliações individuais

Nome do discente	**Nota (0-20)**
Simão Gabriel Mazive	**20**
Inercio Bélton Fanequiço	**20**
Frank Salvador Cumaio	**20**

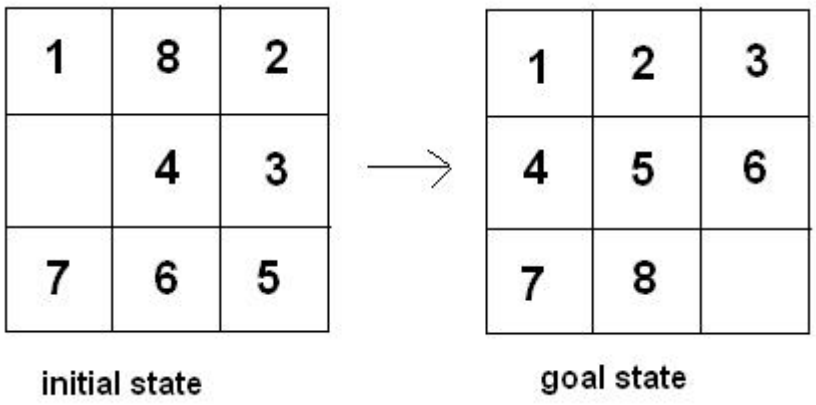
Abril, 2017

Índice

1. Descrição do problema

O jogo do 8-Puzzle é um jogo de tabuleiro de blocos deslizáveis, que consiste em um tabuleiro quadrado com nove divisões (matriz), contendo números (ou letras) e um espaço vazio. Esse espaço vazio permite que as peças vizinhas possam "deslizar", ou seja, uma peça ao lado de um espaço vazio pode ser movida para essa lacuna, criando uma nova posição de lacuna. Em outras palavras, a lacuna pode ser trocada de posição com uma peça adjacente (horizontal e verticalmente). Dessa maneira pode-se mudar a configuração inicial da matriz, movimentando-se as peças. A meta do jogo é chegar a uma certa configuração final (estado goal).

Ex:



Portanto, o jogo consiste em partir de uma configuração inicial, movimentar as peças com o objectivo de chegar a configuração final (goal).

O presente trabalho consiste em desenvolver um programa em JavaScript que resolva o 8-Puzzle usando o algoritmo A*.

2. Algoritmo A* (A asterisco)

O A* é um algoritmo de busca em grafos que encontra um caminho entre um dado nó de origem e um dado nó de destino. Ele emprega uma estimativa heurística que classifica cada nó pela estimativa de melhor rota, passando por esse nó, até o nó destino. É garantido que o A* sempre encontra um caminho entre origem e destino, caso haja um, e mais que isso, ele sempre encontra o caminho com menor custo. Para isso utiliza as seguintes funções:

- g(n) = função de avaliação do custo para chegar até o nó (soma dos custos da aplicação de cada regra que leva do estado inicial até este nó.)
- h(n) = função de avaliação que estima o custo para se chegar deste nó até o estado final. Esta função é a responsável pela parte heurística do processamento.

- f(n) = g(n) + h(n) – É o custo estimado da solução mais barata.

Segundo (Costa & Simões, 2008) o algoritmo possui a seguinte descrição:

Funcao A*(problema, insereListaOrdenada,g+h): solução ou falha

1. I_nos ← FazListaOrdenada(Estadoinicial(problema))
2. **Repete**
   - 2.1. **Se** vaziaListaOrdenada(I_nos) **entao**
     - 2.1.1. **Devolve** falha

     Fim_de_se
   - 2.2. **no** ← RetiraListaOrdenada(I_nos)
   - 2.3. **se** TesteObjectivo(no) **Entao**
     - 2.3.1. **Devolve** no

   senao
     - 2.3.2. **InsereListaOrdenada(I_nos,g+h(Expansao(no,Operadores,(Problema))))**

     Fim_de_se

   Fim_de_repete

Fim de funcao.

2.1. Características do A*

Algoritmo completo.

Segundo (Costa & Simões, 2008) o A* é completo, pois ele vai expandindo e analisando nós por valores crescentes de f. A única forma deste algoritmo não encontrar uma solução é existir uma infinidade de nós com um valor f inferior ao valor f da solução.

Algoritmo óptimo.

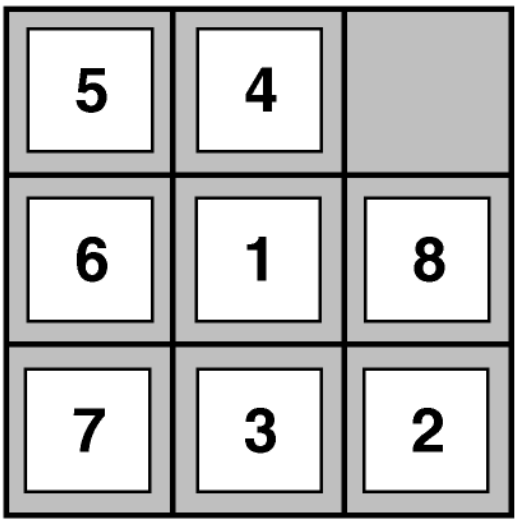
Segundo (Costa & Simões, 2008) o A* é óptimo, pois discrimina entre as várias soluções possíveis encontrando sempre a melhor.

2.2. Heurísticas Admissíveis (para o caso 8 puzzle)

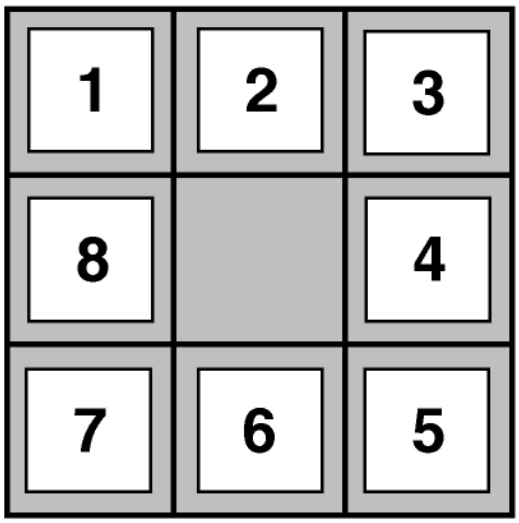
Heurísticas são processos cognitivos empregues em decisões não racionais, sendo definidas como estratégias que ignoram parte da informação com o objetivo de tornar a escolha mais fácil e rápida. Para o caso do nosso trabalho iremos usar as seguintes heurísticas:

- Número de ladrilhos fora de lugar - h1(n), verificando quantos elementos encontram-se fora do seu ladrilho para formar o estado goal.
- Manhattan distance - h2(n), faz-se a soma total de número de ladrilhos da posição actual a posição goal de cada elemento.

Ex:



Start State



Goal State

h1(s) = 7

h2(s) = 2+3+3+2+4+2+0+2 = 18

3. Solução do Problema

De modo a solucionar o problema descrito acima (8 Puzzle) com a implementação do algoritmo A* optamos pela linguagem de programação JavaScript, que é uma linguagem de programação de alto nível e dinâmica usada em ambiente web.

Foi concedida uma página web com intuito de facilitar a apresentação dos resultados.

Para a resolução do problema definiu se o seguinte estado como o estado final (goal):

1	2	3
4	5	6
7	8	

Onde o espaço em branco é representado pelo número 9.

4. Código Fonte do programa

4.1. Definição do estado objectivo

```
var meta =[ ["1","2","3"],    // estado objectivo
            ["4","5","6"],
            ["7","8","9"] ];
```

4.2. Método para Buscar a solução

```
// seleciona o método e inicia a busca
function buscaSolucao() {
	var modo = document.getElementById("algoritmo").value;
	if (!modo)
		return;
	ultimo = copiaEstado(estado); // salva o estado atual do tabuleiro
	desativaBotoes();
	tempoInicio = new Date().getTime();
	pilha = [];
	fechados = [];
	solucao = [];
	nodos = 0;
	movimentos = 0;   // zera movimentos feitos pelo jogador
	if (modo == "BAI")
		aprofundamentoIterativo(0);
	else if (modo[0] == "A") {    // heurísticas do A*
		var nodo = {estado: estado, profundidade: 0, pai: null, valorf:
0, valorg: 0, valorh: 0};
		nodo.valorh = calculaHeuristica(estado,modo);
		nodo.valorf = nodo.valorh;
		pilha.push(nodo);
		document.getElementById("aprofund").innerHTML = profundMax;
		iteracaoBusca(modo,profundMax);
	}
	else {
		var nodo = {estado: estado, profundidade: 0, pai: null};
		pilha.push(nodo);
		document.getElementById("aprofund").innerHTML = profundMax;
		iteracaoBusca(modo,profundMax);
	}
}
```

4.3. Função para calcular heurística utilizada pelo A*

```
function calculaHeuristica(estado,modo) {
	var x,y,n,d,py,px;
	n = 0;
	for (y=0; y<3; y++)
		for(x=0; x<3; x++) {
			if (estado[y][x] != meta[y][x] && estado[y][x] != "9") {
	// verifica peças fora do lugar, sem contar o espaço em branco
```

```
                        if (modo == "A1") // heurística 1 - apenas conta
quantas peças estão fora do lugar
                              n++;
      else if (modo == "A2") { // heurística 2 - calcula distância total
das peças de suas posições corretas
          py = parseInt((estado[y][x]-1)/3);    // calcula linha correta,
baseado no valor da peça
            px = estado[y][x]-py*3-1;  // calcula coluna correta
            d = Math.abs(x-px) + Math.abs(y-py); // Manhattan Distance
                              n += d;
                        }
                        else {
                        alert("Erro na chamada - heurística inválida!");
                              exit;
                        }
                  }
            }
      return n;
}
```

4.4. Casos de Teste

```
function casosTeste(n) {
  switch (n) {
     case 1:
        estado = [["7","1","3"],["2","6","9"],["5","4","8"]];  // Custo ate
a solucao f(n) =  11
        break;
     case 2:
        estado=[["2","3","4"],["1","9","5"],["8","7","6"]];
// Custo ate a solucao f(n) = 16
        break;
     case 3:
        estado  =  [["2","3","7"],["5","4","8"],["9","6","1"]];    //  f(n)
Custo ate a solucao  26
        break;
  }
  exibeEstado(estado);
```

6. Referências Bibliográficas

Costa, E. & Simões, A., 2008. *Inteligência Artificial - Fundamentos e Aplicações.* 2ª edição ed. s.l.:s.n.

Princípios de Desenvolvimento de Algoritmos EP no. 2, acedido em http://www.vision.ime.usp.br/~pmiranda/mac122_2s11/EPs/EP2.pdf, [Acedido a 1/04/2017]

código-fonte consultado em: http://henriquevianna.com/code/ia/8puzzle.html [Acedido a 29/03/17]