8-puzzle

Equipe: Guilherme Assing Vieira e Gabriel Saldanha

Para executar:

Abrir arquivo index.html e ver o console no browser.

Para alterar a heurística, comentar ou descomentar as linhas no começo do arquivo tudo.js.

Para mudar o puzzle que vai ser executado, na linha 176 do arquivo tudo.js ou criar outro.

Representação do estado

O estado é tem a estrutura de uma matriz, ou seja, um array de arrays:

```
let puzzle = new Puzzle([
    [1,2,3],
    [4,5,6],
    [7,8,9]
])
```

Como auxiliar de navegação e de decisão a partir da heurística e do tamanho do caminho temos um objeto no seguinte formato:

```
{
    puzzle, //Objeto Puzzle
    heuristic, //Para manter estado da heuristica
    parent, //Nodo de qual esse foi originado
    final, //Heuristica + tamanho do caminho
    pathSize //Tamanho do caminho
}
```

Estrutura para fronteira e nodos fechados

A estrutura é basicamente uma lista de nodos, os quais são conectados entre si a partir do campo 'parent' em cada nodo.

Descrição da implementação das heurísticas

Distancia até o lugar correto

A heurística é calculada baseada na soma das distancias de todos os números até o lugar correto sem andar pelas diagonais.

```
function distanceToRightPlace(number, i, j) {
    var { x, y } = solvedPuzzle.getPosition(number);
    var distance = Math.abs(i - x) + Math.abs(j - y);
    if (!number)
        distance = 0;
    return { distance, number };
}
```

Quantidade no lugar errado

Basicamente, se o número está no lugar errado, soma 1 à heuristica.

```
function isAtRightPlace(number, i, j) {
    var { x, y } = solvedPuzzle.getPosition(number);
    return x == i && j == y;
}
...
    sum += isAtRightPlace(number, i, j) ? 0 : 1;
...
```

Como foi gerenciada a fronteira, verificações, quais etapas foram feitas ao adicionar um estado na fronteira

(explicação das estratégias, respectivos métodos e possibilidades além do que foi implementado)

A fronteira é inicializada com apenas o nodo inicial, e um loop é executado para encontrar o caminho mais curto. É feito um reduce na fronteira para identificar o nodo com heuristica + tamanho do caminho menor, o qual será utilizado para continuar com o algorítmo.

```
let lower = openNodes.reduce((prev, curr) => prev.final < curr.final ? prev : curr)
closedNodes.push(lower);
openNodes = openNodes.filter(i => i != lower);
```

A partir desse nodo de menor h + t, é retornado todas as matrizes possiveis a partir desse nodo e são adicionadas na fronteira apenas as que não estão na lista de nodos fechados.

```
let puzzles = lower.puzzle.getAllPossibleMatrixes();
for (let m of puzzles) {
    if (closedNodes.filter(o => isEqual(o.puzzle.matrix, m.matrix)).length == 0) {
        let obj = { puzzle: m, heuristic: heuristic(m), parent: lower };
        obj.pathSize = getPathSize(obj);
        obj.final = obj.pathSize + obj.heuristic;
        openNodes.push(obj)
    }
}
```

O papel de cada classe e os métodos principais

run.js

A classe principal da execução de algoritmos com heurística. Ela que tem toda a lógica da gestão de fronteira, nodos fechados e exibição de dados em tela.

matrixes.js

Guarda apenas a matriz resolvida e exporta a matriz que será utilizada na busca.

heuristics/quantityAtWrongPlace

Arquivo contendo a lógica da heurística, com um método em si para retornar a heurística baseada no número e no local onde o mesmo se econtra e outro método que percorre a matriz e soma os valores.

heuristics/distanceToRightPlace

Arquivo contendo a lógica da heurística de distancia até o lugar correto. O método 'isAtRightPlace' retorna true ou false, checando se as posições dos números coincidem ou não.

heuristics/both

Uma heurística que utiliza ambas as anteriores.

puzzle.js

A classe onde se concentra todos os métodos de navegação pelo puzzle, desde movimentação, checagem de posições dos números e retorno da posição tendo como input um número.

Questão 7

Função de utilidade

Como função de utilidade, um estado final pode ter 3 possibiliades diferentes:

- Empate: 0
- Vitória PC: 1
- Vitória Player: -1

Função heurística:

A partir do estado, para cada peça do usuário, verificar quantas estão conectadas na verticao e na horizontal e somar na heurística. E para cada peça do computador, fazer a mesma verificação e subtrair da heurística, como exemplifica o seguinte codigo:

```
let matrix; // Matriz com os valores, -1 é usuário, 0 é vazio e +1 é o computador.
let heuristic = 0; // Heurística para o computador, quanto mais próximo de 0 melhor para o pc
for (let i = 0; i < matrix.length; i++){
    for (let j = 0; j < matrix[i].lenght; j++){
        if (matrix[i][j] == 1){
            heuristic -= heuristic(matrix, i, j);
        }
        if (matrix[i][j] == -1){
            heuristic += heuristic(matrix, i, j);
        }
    }
}
// Quantidade de peças juntas conectadas verticalmente e horizontalmente
function heuristic(matrix, i, j){
    ...
}
```

Heurística com exemplo:

```
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0,-1, 0, 0]
[0, 0, 0, 0, 1, 0, 0]
[0, 0, 0,-1, 1, 0, 0]
```

O primeiro -1 de cima pra baixo não tem nenhum -1 ao lado: soma nada

Primeiro 1 de cima pra baixo tem um 1 conectado: subtrai 1

Segundo 1 também está concetado com um 1: Subtrai 1

-1 da linha de baixo não está concetado a nenhum -1: Soma nada

Resultado: -2