

Meteor Challenge

Autor: Artur Augusto Rocha Coelho

Esse documento possui o raciocínio que utilizei para resolver o desafio proposto pela empresa Tarken. O código utilizado pode ser encontrado no seguinte repositório:

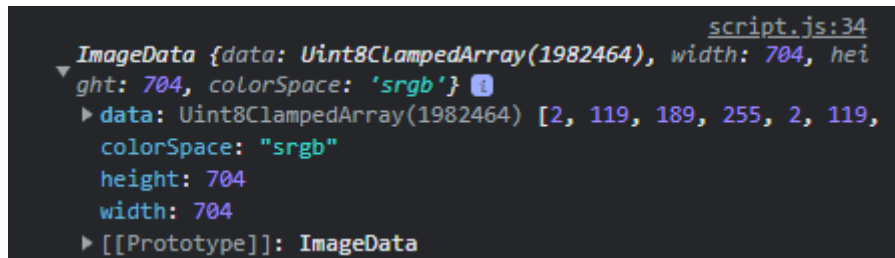
<https://github.com/artur-coelho/meteor>

Respostas encontradas:

Number of Stars	315
Number of Meteors	328
Meteors falling on the Water	105
(optional) Hidden Phrase	Not found

Resoluções:

Para resolver esse desafio, eu utilizei do objeto `ImageData`, através dele foi possível extrair as informações dos pixels que formam a imagem em questão. Ele fornece, além das informações de altura e largura, um array unidimensional contendo os dados ordenados em formato RGBA, com valores inteiros que variam de 0 a 255.



```
script.js:34
ImageData {data: Uint8ClampedArray(1982464), width: 704, height: 704, colorSpace: 'srgb'}
  ▶ data: Uint8ClampedArray(1982464) [2, 119, 189, 255, 2, 119, ...]
    colorSpace: "srgb"
    height: 704
    width: 704
  ▶ [[Prototype]]: ImageData
```

Dessa forma, para avaliar a cor de determinado pixel de acordo sua posição “i” no array, bastava acessar a informação sabendo que:

`data[i]` : Representa o espectro “R” de “red”
`data[i + 1]` : Representa o espectro “G” de “green”
`data[i + 2]` : Representa o espectro “B” de “blue”
`data[i + 3]` : Representa o espectro “A” de “alfa”

E as cores que me interessavam para a resolução desse desafio eram branco, vermelho e azul, os quais possuem a seguinte representação no padrão RGB:

Branco: (255, 255, 255)

Vermelho: (255, 0, 0)

Azul: (0, 0, 255)

Para a resolução dos itens 1 e 2 desse desafio, eu criei um código que itera sobre o array `data` e verifica as informações de cada pixel, se ele for vermelho ou azul, incrementava o contador de suas respectivas cores:

```
for (let i = 0; i < data.length; i += 4) {  
  if (data[i] === 255 && data[i + 1] === 255 && data[i + 2] === 255) {  
    whitePixelsAmount += 1;  
    let whitePixelColumnPosition = (i / 4) % imageWidth;  
    whitePixelsColumnPositions.push(whitePixelColumnPosition);  
  } else if (data[i] === 255 && data[i + 1] === 0 && data[i + 2] === 0) {  
    redPixelsAmount += 1;  
    let redPixelColumnPosition = (i / 4) % imageWidth;  
    redPixelsColumnPositions.push(redPixelColumnPosition);  
  }  
}
```

Para a resolução do item 3, foi necessário também durante a iteração no array, gerar e guardar a posição de sua coluna na matriz que representa a imagem, para isso eu utilizei o seguinte cálculo:

Posição da coluna = $(i / 4) \% \text{width}$,

onde “i” representa a posição no array e “width” a largura da imagem

Aqui eu divido “i” por quatro pois um pixel é representado a cada quatro índices no array, e através do resto da divisão desse valor pela quantidade de colunas (largura), é possível determinar em qual coluna ela se encontra.

Sendo assim, o código ao se deparar com um pixel vermelho, armazena a posição de sua coluna. Para os pixels azuis, eu apenas armazeno a posições das colunas na primeira linha da imagem em que eles aparecem, pois os que estão abaixo delas não são úteis para a resolução desse problema:

```

} else if (data[i] === 255 && data[i + 1] === 0 && data[i + 2] === 0) {
  redPixelsAmount += 1;
  let redPixelColumnPosition = (i / 4) % imageWidth;
  redPixelsColumnPositions.push(redPixelColumnPosition);
} else if (data[i] === 0 && data[i + 1] === 0 && data[i + 2] === 255) {
  const actualRow = Math.floor(i / 4 / imageWidth);

  if (firstWaterRow === null) {
    firstWaterRow = actualRow;
  }

  if (actualRow === firstWaterRow) {
    let bluePixelColumnPosition = (i / 4) % imageWidth;
    bluePixelsColumnPositions.push(bluePixelColumnPosition);
  }
}
}

```

Com essas informações armazenadas, eu itero sobre o array de pixels vermelhos(meteoros) e comparo com os pixels azuis(água), caso estejam na mesma coluna, então significa que o meteoro irá cair na água:

```

63 let meteorsOnWaterAmount = 0;
64 const meteorsOnWaterCounter = () => {
65   redPixelsColumnPositions.map((redColumnPosition) => {
66     bluePixelsColumnPositions.map((blueColumnPosition) => {
67       if (redColumnPosition === blueColumnPosition) {
68         meteorsOnWaterAmount += 1;
69       }
70     });
71   });
72 };

```

Para a resolução do item 4, foram tentadas diversas abordagens, primeiro apliquei os filtros de imagem mais utilizados e também gerei uma imagem com apenas os pixels considerados relevantes, na esperança de alguma mensagem aparecer, mas não tive sucesso:



☐ Original ☒ Grayscale ☐ Inverted ☐ removeBlank



☐ Original ☐ Grayscale ☒ Inverted ☐ removeBlank



☐ Original ☐ Grayscale ☐ Inverted ☒ removeBlank

Também tentei gerar um número em binário, ordenado de acordo com a posição dos astros no array, onde os meteoros representavam o 1 e as estrelas o 0:

```
0000000111100001111111111101111000011111100000011111110111 script.js:21
11100000011111111111001110011000111100011111100000001111111011100111111111
11111110110111110000000001111111100100001111111111100000111111110111111
11111111110011110111111111110111110001111101011111001111100001111111011
111111111111100011111111111100001111000011111111100111011111100111111111
11011011111110000011111101111111000000011110011111110100000000110011000
00011110110001000011000000110000000000000010000001100000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000
```

Mas também não foi possível extrair nenhuma informação disso, até porque não daria para extrair 175 caracteres disso, como foi dito na dica.

Uma das últimas ideias que tive consistia em pegar a posição da coluna de cada estrela ou meteoro e fazer o resto da sua divisão por 127, na esperança de que suas respectivas posições na tabela ASCII gerassem uma frase. Mas essa tentativa também fugiria do limite de 175 caracteres que foi dado.

```
whitePixelsColumnPositions script.js:16
(315) [219, 194, 137, 295, 74, 81, 242, 617, 146, 455, 89, 639, 276, 383, 5
62, 337, 97, 147, 329, 663, 260, 374, 515, 379, 479, 503, 204, 694, 532, 40
9, 645, 207, 678, 26, 9, 642, 241, 682, 354, 54, 442, 574, 369, 372, 94, 60
▶ 9, 341, 326, 435, 613, 517, 370, 262, 572, 189, 250, 285, 339, 547, 679, 63
6, 653, 87, 132, 619, 186, 151, 138, 150, 484, 29, 382, 615, 2, 82, 106, 59
8, 569, 266, 314, 234, 561, 650, 293, 302, 441, 393, 236, 214, 433, 537, 21
8, 418, 693, 115, 58, 322, 421, 177, 34, ...]

redPixelsColumnPositions script.js:17
(328) [210, 253, 375, 513, 70, 386, 557, 577, 9, 69, 290, 434, 459, 681, 30
8, 314, 491, 217, 495, 541, 669, 78, 287, 354, 373, 406, 484, 492, 66, 95,
177, 233, 234, 298, 387, 674, 686, 318, 322, 365, 487, 556, 567, 309, 129,
▶ 147, 193, 374, 385, 534, 6, 33, 209, 466, 685, 242, 289, 342, 442, 647, 26,
326, 525, 684, 75, 114, 321, 137, 215, 330, 519, 113, 313, 170, 195, 279, 3
05, 340, 390, 282, 551, 606, 2, 3, 97, 378, 382, 543, 687, 470, 524, 569, 5
70, 658, 42, 130, 162, 214, 361, 529, ...]
```

Além das citadas, também tentei outras abordagens que infelizmente não me trouxeram resultado, por isso não foi possível realizar o item 4.

Documentação usada como referência:

[https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/
Pixel_manipulation_with_canvas](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas)