**Desafio do Meteoro**

***Por Alec Can Yalçin***

Tarefas:

1. *Contar o número de estrelas*
2. *Contar o número de meteoros*
3. *Considerando que os meteoros estão caindo perpendicularmente ao solo (nível da água), conte quantos irão cair na água.*
4. *(opcional) Encontre a frase escondida nos pontos entre o céu.*
   1. *DICA 1: 175 Caracteres*
   2. *DICA 2: A maioria dos códigos das últimas tarefas podem ser reusados.*

**Respostas Simples**

|  |  |
| --- | --- |
| Número de Estrelas | 315 |
| Número de Meteoros | 328 |
| Meteoros caindo na Água | 105 |
| (Opcional) Frase escondida | -- |

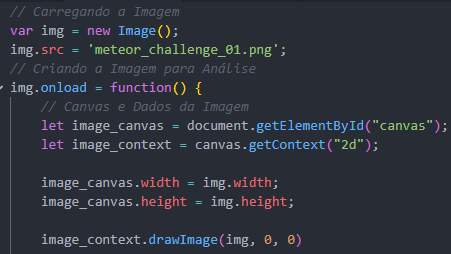
**Metodologia Utilizada**

Para concluir o desafio, foi necessário a análise de quais possibilidades eram possíveis para resolver as tarefas. O primeiro obstáculo foi encontrar a maneira de resolver o problema, mas tudo parte de uma simples premissa: conseguir identificar os pixels na imagem. Tendo o conhecimento de que as imagens, em resumo, são uma matriz de tamanho “*n x m”*, onde *“n”* é a altura e ”*m”* a largura em pixels, foi essencial escolher uma linguagem de programação que estivesse de acordo com a facilidade da análise.

Com uma simples análise, a partir da facilidade de imagem e também do perfil que a Tarken busca com programação, a linguagem escolhida foi *Javascript.* Com a seguinte proposta:

1. Por ser uma linguagem já instalada automaticamente, qualquer que fosse o recrutador pode ter acesso as respostas tendo um navegador browser disponível;
2. Por ser uma linguagem desenvolvida para a web, ela conta com uma vasta biblioteca já inclusa em sua base de código, que não necessita o *download* de nenhum arquivo adicional;
3. Mundialmente conhecida por sua facilidade, Javascript consegue resumir grandes problemas em pequenos scripts, sendo fácil de ser analisada.

Com a listagem das vantagens do Javascript para esse desafio, partiremos agora para a análise do código. Tendo em mente a ideia de transformar a imagem em uma matriz de duas dimensões, o passo seguinte foi utilizar-se do *object Image* para criar um *canvas* que corresponde-se a imagem original. A vantagem de utilizar *canvas* ao invés da imagem comum como forma de coletar os dados se dá pelas características do objeto *canvas*. Em síntese, sua ideia é ser como uma tela que pode ser pintada ou modificada a maneira que for preciso, então todas as ferramentas para traduzir imagens para canvas fazem com que o canvas ganhe a característica dessas imagens.



Com isso em mente, o código acima descreve os passos necessário para transformar a imagem original em uma canvas. Note que a função *onload* serve para que essa transformação aconteça de maneira assíncrona, sendo necessário que todo o código adiante (que altere o canvas) precise ser feito dentro dele ou que espere ele ser terminado. Portanto, o restante da lógica do código vai se encaminhar dentro dessa função, mas fará chamadas de outras funções quando necessário.

Para resolver agora os desafios, é preciso pegar esse canvas e traduzi-lo em uma matriz n x m que tenha os valores dos pixels, as cores, guardados. É possível fazê-lo sem muita dificuldade como dito antes, pois o canvas possui uma função chamada *getImageData* que nos permite resgatar justamente essas informações. A função *imageArray* foi criada para resgatar esses dados e transformá-los na matriz desejada.



Por fim, temos o vetor com todos os valores de cores e pixels da imagem, separados por linhas e colunas. Essa separação de linhas e colunas vai se tornar essencial para resolver todos os desafios propostos. Note que temos um aumento de 4 em 4, isso se deve ao fato que cada pixel carrega a informação das cores, que estão organizadas em um padrão *rgba*, esse padrão guarda valores para *red, green, blue* e *alpha.* Assim, salvamos cada uma delas para cada linha e coluna que se encontram.

Agora, para resolvermos definitivamente as tarefas 1 e 2 precisamos encontrar os pixels de referência para as Estrelas e Meteoros.

Referência de Pixel

*(pure White): Stars*

*(pure red): Meteors*

*(pure blue): Water*

*(pure black): Ground*

Com a indicação das cores, podemos ver cada cor na imagem possui um significado. Os pontos brancos são as estrelas, pontos vermelhos meteoros, pontos azuis água e pontos pretos o chão. É bem explicito a ideia de “pure [color]”, o que corresponde com o resultado da matriz que nos foi retornada. Nela há quatro números que representam o *rgba* dos pixels da imagem, considerando a “pureza” descrita, em rgba ela se dá de números de 0 a 255, e para encontrarmos cores puras basta que o rgb estejam um em 255 e o restante em 0, ou todos com o mesmo valor 0 ou 255. Olhemos a equivalência

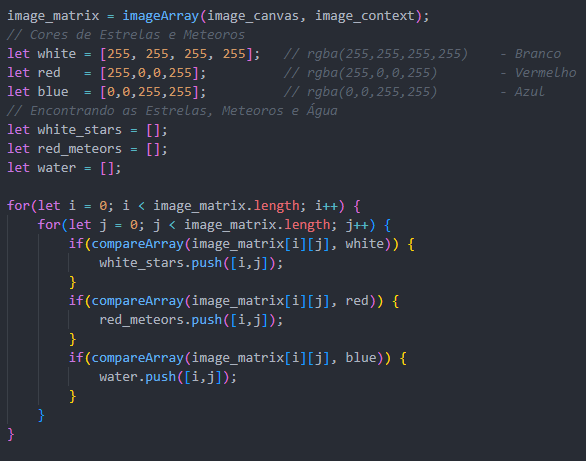
Referência de Cor

*(pure White): Stars -> rgba(255,255,255,255)*

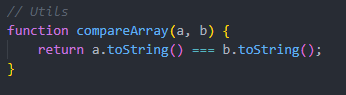
*(pure red): Meteors -> rgba(255,0,0,255)*

*(pure blue): Water -> rgba(0,0,255,255)*

*(pure black): Ground -> rgba(0,0,0,0)*



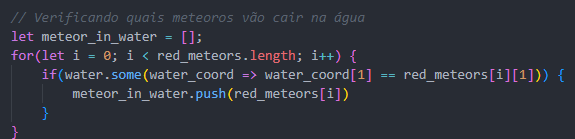
Com isso, podemos achar as estrelas, os meteoros e qualquer um dos elementos citados que tenham esses valores. Percorrendo rapidamente a matriz criada, podemos comparar os valores delas com o rgba descrito acima. Para isso, foi utilizada uma função para comparar esses valores por serem *arrays*.



Com isso, a resposta do desafio 1 e 2 foram encontradas com facilidade. Com uma visão mais atenta, nota-se que eu também percorri os valores da “água”, e isso tem uma motivação: Serão usados para encontrar os meteoros que se encontram perpendicularmente a água.

Para realizar esse feito, é necessário entender que quando algo está “perpendicular” em um plano 2d é estritamente verdadeiro que ele está a 90° do que se está sendo comparado, ou seja, ele está ou na horizontal ou vertical de acordo com o valor comparado. Se considerarmos que os meteoros estão em planos horizontais, eles estão perpendicularmente no plano vertical em comparação com a água. Sendo assim, a única evidência de encontro entre eles é a própria altura que se encontram.

Sabendo-se disso, para verificar se estão de fato perpendiculares a altura é o indicador de posição, sendo irrelevante. Se o meteoro está a 100 px, 200 px, 700px de distância não importa, o que importa é se ele está na mesma coluna. Ou seja, a posição da linha que indica a coluna devem ser as mesmas para ambos. E como os valores das águas foram salvos, suas coordenadas x e y estão guardadas, se compararmos a coordenada x dos meteoros e da água e elas baterem, então estão perpendiculares.



A função “some” percorre a matriz water, enquanto o for percorre a matriz dos meteoros. Em seguida, uma comparação é feita. Como os valores em cada vetor estão guardados respectivamente como: [height, width] então o valor 0 desses vetores se referem ao plano y e o valor 1 se refere ao plano x. Comparando o plano x de cada uma na posição [1] de seus vetores, podemos salvar quais estão localizados na mesma coluna.