Meteor Challenge

Tasks:

1. Count the number of Stars
2. Count the number of Meteors
3. If the Meteors are falling perpendicularly to the Ground (Water level), count how many will fall on the Water
4. (optional) Find the phrase that is hidden in the dots in the sky.
    a. HINT 1: 177 Characters
    b. HINT 2: Most of the last tasks' code can be reused for this one

Please, send us the result and code you used to solve the tasks above. Explain how you achieved the results in each question. Good work!!

Subject: [CHALLENGE] [METEOR] *your name*

[Sample] Answers:

Number of Stars	315
Number of Meteors	328
Meteors falling on the Water	105
(optional) Hidden Phrase	

Pixel Ref:

(pure white) Stars
(pure red) Meteors
(pure blue) Water
(pure black) Ground

Resolução (por Gabriela Melo Silva)

Antes de partir para a resolução do desafio em si, primeiramente, procurei alguma biblioteca que me auxiliasse com o processamento da imagem a ser analisada. Resolvi então utilizar a biblioteca OpenCV, com auxílio da linguagem Python. Logo, é necessário ter os dois instalados na máquina antes de executar o programa.

Após análise visual, percebi que cada estrela e meteoro, na verdade, se trata de um único pixel da imagem, puramente vermelho ou puramente branco, como é dito no desafio. Logo, a minha ideia inicial para as duas primeiras tarefas foi percorrer todos os pixels da imagem e verificar se a cor dele corresponde a branco

ou vermelho, e para cada correspondência, incrementar um contador de estrelas ou meteoros.

Tarefa 1 - Contar o número de estrelas

Inicialmente, utilizei a função `cv.imread()` para ler a imagem necessária para o desafio, armazenando-a numa variável chamada `img`.

Para contar o número de estrelas, primeiramente, armazenei o valor RGB do branco em uma tupla `white = (255, 255, 255)` e comecei a percorrer a imagem com o auxílio de dois laços for, sendo o mais externo `for i in range (0, img.shape[0])`, e o interno `for j in range (0, img.shape[1])`. O `.shape` retorna uma tupla contendo o número de linhas, colunas e canais da imagem, logo, `img.shape[0]` representa as linhas, e `img.shape[1]` representa as colunas.

Dentro do laço mais interno, para comparar os pixels de cada coluna desde a linha 0 até a última linha, inseri um `if tuple(img[j, i]) == white: stars += 1`, assim, cada vez que uma estrela fosse encontrada, a variável `stars` (iniciada em 0) adicionaria uma estrela. O `[j, i]` representa justamente a linha e coluna do pixel atual.

Tarefa 2 - Contar o número de meteoros

A segunda tarefa foi resolvida de maneira análoga à primeira, mas agora, a comparação dentro dos laços se deu com uma nova variável. Para isso, precisei me atentar ao fato de que a biblioteca OpenCV utiliza o formato BGR ao invés de RGB, logo, o vermelho puro seria (0, 0, 255) ao invés de (255, 0, 0). Portanto, criei a variável `red = (0, 0, 255)`, e utilizei em um outro if dentro do laço para incrementar um novo contador chamado `meteors` toda vez que um pixel vermelho fosse encontrado.

Tarefa 3 - Se os meteoros estão caindo perpendicularmente ao chão (nível da água), contar quantos vão cair na água

Esta tarefa foi o motivo da escolha de percorrer a imagem coluna por coluna ao invés de linha por linha (o que foi feito através do uso de `img[j, i]` na comparação ao invés de `img[i, j]`). Para resolvê-la, precisei alterar apenas um pouco o código resultante após o término da tarefa 2, de forma que, ao invés de incrementar diretamente a `meteors` no laço, criar uma variável temporária chamada `temp_meteors` para armazenar os meteoros de cada coluna, e apenas depois incrementar.

Além disso, também foram criadas as variáveis `blue = (255, 0, 0)`, que representa o azul puro no formato BGR, `is_water`, para servir como uma espécie de *flag* para verificar se há água naquela coluna, e `meteors_in_water`, para incrementar o número de meteoros da variável temporária apenas se houver água naquela coluna.

Portanto, foi criado um novo `if` dentro dos dois laços, para dessa vez verificar se há um pixel azul naquela coluna, e se houver, trocar a variável `is_water` para 1 e quebrar o laço (já que abaixo da água só pode haver mais água ou terra, o que é irrelevante para os fins do programa).

Fora do laço interno, é feita também uma verificação para saber se houve água naquela coluna, e se houver, a variável `temp_meteors` é incrementada na `meteors_in_water`, e além disso, ela também é incrementada na variável `meteors` (sem nenhuma verificação necessária, já que o objetivo do desafio 2 é saber o total de meteoros). Após isso, tanto a `temp_meteors` quanto a `is_water` são zeradas para a próxima verificação do laço interno.

Após a execução do código, foi encontrado o seguinte resultado (já preenchido na tabela):

```
Stars: 315
Meteors: 328
```

Meteors in water: 105

Tarefa 4 - Encontrar a frase escondida nos pontos no céu

Inicialmente, decidi verificar se havia mais de uma estrela ou meteoro por linha, inserindo um contador em cada linha para as estrelas e os meteoros. Porém, vi que cada linha tem uma quantidade de 0 ou 1 meteoros/estrelas.

Depois, apenas para fins de comparações futuras, resolvi verificar quantas colunas possuem água, utilizando uma variável como flag para identificar toda vez que água é encontrada em uma coluna, e outra variável para verificar a flag e incrementar seu valor. Assim, cheguei no valor de 221 colunas com água no total.

Então, decidi verificar quantas colunas possuíam uma estrela e um meteoro ao mesmo tempo, chegando em 205.

Nesse ponto, parei para pensar se na verdade não deveria trabalhar com as linhas da imagem ao invés das colunas, e assim, resolvi realizar novas verificações.

Pensei então que, já que em cada coluna há apenas uma estrela, um meteoro, os dois ou nenhum, talvez tivesse que ter algum código binário como resultado para então traduzir para a frase. Transformando cada coluna que possui apenas uma estrela ou meteoro em 0, cada coluna que possui os dois em 1, e cada coluna que não possui nenhum em um espaço, acabei com o seguinte código:

```
       1   1  1  0  1 110 0    1  010 110  00  1      01 000
10 1000 110 1    1      01    0 11   0  11 0101 110 0 1 110 0
1       11 1    11 1111 111 101  1      11 0    11    1 110  1
01  0    1       1100  1 11 1011 110 1 1  1       11 0    11 1
0 110 1    1 000   1       1  0  1 110 1    1  000 010  00  1
11     1 11    1  11 1001 110 1 0 110 1    1       11 0    11
1111 111 010  1        11 0    11    0 110  0  01  0     1
0100  0 01 1101 010 0 0  0       11   11 11    1 11 101   1
01  000 10  0 0 110 0    1       11 0    11 1  1 110 0    1
10    1 11 100  11  1    1       11 0 01 11  0 1 11  1 0 110
```

1 01 00 0 10 0011 111 10 11 0 1 11 110 11 100 1 11 11 11 0101

Formatando os números:

0011 0001 0000 0001 0110 0000 0001 0010 0110 0000 0001 0001
0000 0010 1000 0110 0001 0001 0001 0000 0011 0000 0011 0101
0110 0000 0001 0110 0000 0001 0011 0001 0011 1111 0111 0101
0001 0011 0000 0011 0001 0110 0001 0001 0000 0001 1100 0001
0011 1011 0110 0001 0001 0001 0011 0000 0011 0001 0000 0110
0001 0001 0000 0001 0001 0000 0001 0110 0001 0001 0000 0010
0000 0001 0011 0001 0011 0001 0011 1001 0110 0001 0000 0110
0001 0001 0011 0000 0011 1111 0111 0010 0001 0011 0000 0011
0000 0110 0000 0001 0000 0001 0100 0000 0001 1101 0010 0000
0000 0000 0011 0011 0011 0001 0011 0101 0001 0001 0000 0010
0000 0000 0110 0000 0001 0011 0000 0011 0001 0001 0110 0000
0001 0010 0001 0011 0100 0011 0001 0001 0011 0000 0001 0011
0000 0001 0011 0001 0000 0110 0001 0001 0000 0000 0010 0011
0111 0010 0011 0000 0001 0011 0110 0011 0100 0001 0011 0011
0011 0101

Como percebi que os números vão no máximo até 16 em binário, pensei em transformar em hexadecimal e depois tentar converter para ASCII, porém, não encontrei nenhum texto legível. Apenas para fins de registro, a conversão para hexadecimal resultou no seguinte:

3 1 0 1 6 0 1 2 6 0 1 1 0 2 8 6 1 1 1 0 3 0 3 5 6 0 1 6 0 1 3
1 3 F 7 5 1 3 0 3 1 6 1 1 0 1 C 1 3 B 6 1 1 1 3 0 3 1 0 6 1 1
0 1 1 0 1 6 1 1 0 2 0 1 3 1 3 1 3 9 6 1 0 6 1 1 3 0 3 F 7 2 1
3 0 3 0 6 0 1 0 1 4 0 1 D 2 0 0 0 3 3 3 1 3 5 1 1 0 2 0 0 6 0
1 3 0 3 1 1 6 0 1 2 1 3 4 3 1 1 3 0 1 3 0 1 3 1 0 6 1 1 0 0 2
3 7 2 3 0 1 3 6 3 4 1 3 3 3 5

Pensei em transformar cada número em sua letra correspondente ao alfabeto, ou até mesmo usar o código binário não formatado para transformar em morse (tanto 0 sendo ponto e 1 sendo traço quanto o contrário) mas como tem muitas repetições, também não foi encontrada nenhuma frase que faça sentido, apesar da tradução para morse ter resultado em 173 caracteres (bem próximo dos 177 esperados), não é possível encontrar uma frase legível, como podemos ver a seguir:

TTTETGETRGITASNBGTTAEMEMÆGETGETMTMŠOKTMEMTGTAETZTMYGTTTMEMTEGTTSTTETGTTSRITMTMTMXGTEGTTMEMŠORTMEMEGEAETLEAQREEEMMMTMKTASNEEGETMEMTTGETNTMDMTTMEAMETMTEGTAIENŬONMETMGMDTMMMÆ.

Logo, parti para outras abordagens, levando em consideração dessa vez as colunas com água. Tentei verificar o número de colunas com água apenas com meteoros (41), apenas com estrelas (32), com os dois (64) ou nenhum (84), o que somados dão 221 colunas com água no total (como já verificado antes). Porém, mesmo tentando realizar várias operações com os valores encontrados, nada se aproximou da quantidade de caracteres da frase escondida (177).

Minha próxima tentativa foi usar as colunas com água com e sem meteoro para criar códigos em binário, porém também não obtive sucesso em encontrar um texto legível.

Tentei verificar as linhas que possuíam meteoros ao invés das colunas, e ver se conseguia fazer algo com os resultados. Inicialmente, tentei ver a soma de todos os meteoros por linha para transformar em letras do alfabeto de acordo com a ordem, mas, como haviam muitas repetições e os valores eram apenas de 1 até 9, só pelos números resultantes já foi perceptível que não havia como formar palavras.

Mesmo testando variações com binário, morse e até aplicar cifra de césar, não foi possível encontrar nenhuma frase legível.