1. **Para quem estou fazendo esta análise?**

Ao desenvolver um trabalho é importante ter em mente para quem ele está sendo direcionado, de modo que tanto a execução quanto o relatório estejam alinhados para o entendimento de todos. Neste caso é direcionado para uma pessoa do RH e para outra(s) com o perfil técnico. Portanto, a execução desse desafio foi feita caracteristicamente técnica (até mesmo pela própria natureza do desafio) e este relatório foi desenvolvido pensando nos dois perfis.

2. **Brainstorming**

A segunda etapa consistiu em fazer perguntas simples para a idealização do código (isso lembra a criação da Ferramenta Canvas, no qual é possível deixar a ideia bem clara, porém nesse caso serão menos que 9 divisões de perguntas). Este é o momento de deixar a criatividade fluir, e as perguntas foram surgindo:

A coloração importa? ***Ter ou não transparência faz diferença?***

Filtros precisam ser utilizados? ***É relevante analisar o gráfico de cores?***

Os contornos são necessários? ***O tamanho da imagem/pixel é importante?***

O formato precisa ser encaixado em um formato pré definido?

O aspecto no domínio da frequência é relevante?

Com essas perguntas foi possível inferir que seria importante relembrar como funciona o padrão de cores RGB e RGBA.

> RGB é a abreviatura de um sistema de cores aditivas em que o Vermelho (Red), o Verde (Green) e o Azul (Blue) são combinados de várias formas de modo a reproduzir um largo espectro cromático. O propósito principal do sistema RGB é a reprodução de cores em dispositivos eletrônicos como monitores de TV e computador, retroprojetores, scanners e câmeras digitais, assim como na fotografia tradicional. (fonte: wikipédia)

> RGBA é o sistema de cores formado pelas cores vermelho (red), verde (green), azul (blue) e pelo canal alfa. O sistema permite exibir todas as cores do sistema RGB e a utilização da transparência de imagem, artifício amplamente usado em softwares de edição de imagem com camadas. (fonte: wikipédia)

Em outras palavras o RGBA é o RGB com inclusão de uma variável que nos dará a informação de transparência do pixel, quando menor o alfa mais transparente o pixel será, isso ocorre mais comumente para imagens com extensão 'PNG', acredito que todos já baixaram uma imagem na internet e a mesma veio com a parte transparente, sem saber ao usar a imagem ficou de uma forma um tanto quanto inusitada.

Outro ponto importante nesse contexto é o padrão Hexadecimal: ele é um padrão em que se misturam letras de A a F (seis letras) e números de zero a nove. O padrão hexadecimal se divide em três pares de combinações, resultando em seis algarismos alfanuméricos. Já o padrão RGB se divide em uma sequência de três números distintos que variam de zero a 255. Podemos então destacar que RGB tem três sequências de números e Hexadecimal tem três pares de algarismos alfanuméricos. (fonte: serprogramador)

Exemplos de cores:

Cor	**Padrão RGB**	**Padrão hex**
azul	0, 0, 255	#0000FF
vermelho	255, 0, 0	#FF0000
branco	255, 255, 255	#FFFFFF
preto	0, 0, 0	#000000

Tabela 1: Exemplos de cores com seus respectivos padrões RGB e hexadecimal.
Fonte: autor.

O sistema de cores RGB é formado através da luz. Seja através da luz do computador, do celular, da câmara fotográfica, filmadora, etc. Cada ponto (pixel - o menor componente de uma imagem digital) na tela significa que há o sistema RGB. Com a evolução da Física ao longo dos anos, os cientistas perceberam que a luz possui um comportamento similar ao das ondas eletromagnéticas, a luz é uma oscilação e se propaga no vácuo com uma certa variação no tempo (frequência). Podemos associá-la como um exemplo para o som, sem caracterizar muitos detalhes o som é uma vibração mecânica do ar, onde frequências diferentes caracterizariam sons graves e agudos. Assim como o som, as frequências determinam as cores para a luz, para uma determinada faixa de frequências podemos observar as cores, e essa faixa de cores é chamada de espectro de luz visível como mostrado na Figura 1. (fonte: caleidoscópio)

Figura 1: espectro visível de cores. Fonte: explicatorium.

Os limites do espectro visível variam de pessoa para pessoa, mais ou menos, sendo assim, os olhos dos seres humanos têm uma faixa definida, se limitando entre 350 nm a 700 nm dos comprimentos de ondas para a luz visível. (fonte: caleidoscópio).

Cor	Comprimento de onda (nm)	Frequência (THz)
Vermelho	625 a 740	480 a 405
Laranja	590 a 625	510 a 480
Amarelo	565 a 590	530 a 510
Verde	500 a 565	600 a 530
Ciano	485 a 500	620 a 600
Azul	440 a 485	680 a 620
Violeta	380 a 440	790 a 680

Figura 2: faixa de frequência e comprimento de onda para algumas cores do espectro visível. Fonte: Infoescola

3. **Referências (pesquisa e estudo)**

Esta etapa consiste no levantamento de informações sobre imagens - e com isso descobrir como o processamento deve ser feito. O INPE disponibiliza alguns documentos que são relevantes sobre processamento de imagem, classificação de imagens e também sobre a segmentação de imagens. Outro material bem relevante encontrado durante a pesquisa foi um sobre o domínio da frequência feito pelo Flávio Viola (LinkedIn) durante sua formação de mestrado pela UFF.

Pesquisa de melhoramento de imagem para análise, com esta foi possível identificar os três passos para o processamento da imagem: pré-processamento, realce e classificação.

4. **Linguagem de programação**

Para o processamento de imagem pode ser usado com facilidade Python, C# e MATLAB, porém MATLAB é um software pago (e BEM pago), para o C# precisarei instalar o Visual Studio, por exemplo, que é bem pesado, com isso foi escolhido Python, que pode ser usado até a própria IDE deles o que o torna bem mais simples e leve. Além disso, Python está em primeiro lugar na lista de linguagens mais populares - Quanto mais um tutorial de linguagem é pesquisado, mais popular a linguagem é considerada. O Jupyter notebook sera o interpretador utilizado.

5. **Busca por bibliotecas**

Esse é o momento de pesquisar os pacotes visuais que atendem as necessidades do projeto:

- O Módulo Pillow pode ser usado por ser Open.
- O módulo NumPy foi adicionado para as operações matemáticas, sua licença.
- O módulo colection foi adicionado para o uso de dicionário, uma biblioteca padrão do Python.

6. **Lógica inicial**

A primeira impressão que eu tive ao ler o desafio foi que seria interessante o uso de dicionários para contabilizar o número de cada cor, como for dito que as cores:

- da estrela é branco puro, ou seja #FFFFFF ou (255, 255, 255),
- do meteóro é vermelho puro, ou seja #FF0000 ou (255, 0, 0),
- da água é azul puro, ou seja #0000FF ou (0, 0, 255) e
- do solo é preto puro, ou seja #000000 ou (0, 0, 0).

O uso da variável dicionário se dá pela chamada das informações da figura diretamente para ser analisadas, essa análise ocorre com a cada parte do dicionário recebendo informações, comparando se já existe, caso exista já é feito um incremento em uma variável referente a entrada, caso contrário cria-se uma nova posição com a informação nova e o contador recebe um valor. Eu preciso relembrar isso, mas eu acho que essa vai ser uma lógica fácil para fazer a primeira contagem, além de muito útil para futuras conferências, já que pode ocorrer da imagem perder informação durante as manipulações.

Para a parte dos meteoros caindo na água poderia ser feita de uma forma rápida fazendo uma lista das posições de cada meteóro (quanto a linha) e depois comparar com a primeira linha que existe água, fazendo uma comparação sem manipulação nem nada, porém acho que será muito mais útil para futuras manipulações para encontrar as letras, a execução de redução da imagem, já que teremos menos pixels para rastrear (rodar em todas as linhas e colunas) informação. Assim, com a imagem condensada, pode ser que seja possível analisar de forma mais clara uma forma de achar as letras.

A parte opcional que fala das letras me chamou mais a atenção, eu senti um desafio e adoro achar padrões e lógicas. A frase escondida no céu dentro dos pontos pode ter sido escondida de várias formas. Uma das opções mais prováveis seria código morse que é uma coisa que conheço desde quando eu era lobinho/escoteiro, foi uma das primeiras formas de codificação que eu conheci. Se realmente for isso, não é tão difícil, porém, pra fazer esse código eu teria que, digamos assim, “truncar” a imagem pra cima ou pra baixo, pra ter todos os pontos juntos.

Se o resultado desse processo for duas linhas, é bem provável que seja realmente código morse. Caso tenha mais linhas pode ser braille, pode ser algo como os valores posições dos pontos no céu convertido para unicode, porém essas são apenas especulações.

A ideia inicial para pensar nisso é descobrir o número de conjunto de códigos (imaginando o morse), pode-se então contar para ver se é realmente isso, a premissa é que são 177 caracteres também é possível pegar o tamanho total da imagem em pixels e dividir pelo total de caracteres para ter noção de opções que podem ser levantadas.

7. **Montagem do código**

O código foi focado em ser visualmente explicativo, para atender aos dois públicos alvo. Com isso acabou sendo um código computacionalmente pesado pelo uso de demasiada quantidade de laços for, principalmente sobre outros laços for.

A lógica implementada para contagem de estrelas e meteoros foi a de dicionário sendo feito uma função para que pudesse conferir posteriormente se durante a implementação não foi perdida informações, a lógica consiste em uma variável que armazena a contagem de todas as quantidades de cada informação de cor, e foi feito para sair duas ou três informações sendo quantidade de vermelho (#FF0000) e branco (#FFFFFF) tendo ou não a saída da quantidade de azul (#0000FF), esta última para conferência se havia sobrado alguma linha de água durante o processo.

O código foi feito para transformar imagem em array, o que possibilitaria fazer operações matemáticas de forma simples. Para a visualização da saída foi feita uma função que transforma o array em imagem, salva com o nome que é fornecido para a função e abre na tela a figura gerada, essa função foi usada para criar as Figuras 4, 5, 6, 7, 8, 9 e 10.

Foi feita uma função para retirar as partes que não haviam estrelas ou meteoros e que deixasse apenas a primeira linha de água, foi feita uma função para comparar a entrada com cores, esta retorna um valor específico para cada cor, foi feita uma para retornar um array de uma figura com todos os elementos pretos - esta foi feita para resolver uma dificuldades que eu estava tendo quanto a codificação de variável, só quando eu estava tentando implementar a parte que transformava em letra as imagens que eu relembrei que o tipo unicode pode representar oito bits ou dezesseis bits ou mais e não é tão comum a mudança dessa classificação para mim, com isso essa função é lenta mas resolve o problema que eu precisava resolver no momento.

Uma função que transforma a imagem com várias cores diferentes em apenas quatro cores, sendo preto, vermelho, branco e azul, isso serve para uma melhor visualização como é vista pela diferença das Figuras 7 (muitas cores) e 8 (apenas as quatro cores), isso ocorre por criar um grande contraste entre o preto com azul, vermelho e branco, além disso somente as quatro cores relevantes continuam na imagem/array.

A última função criada é uma que serve para fazer o “truncamento” (condensar as informações em menos linhas) da imagem, a sua utilização é vista da diferença da Figura 8 para a Figura 9.

O código chama essas funções e utiliza da redução de linhas sem importância para chegar na Figura 10 (final). Com essa parte foi testado que o vermelho se encontrava na linha do azul, caso estivesse um contador recebia um incremento, assim foi achado o número de meteoros que cairiam na água.

Figura 3 - Figura original recebida.

Figura 4 - Figura após a primeira retirada de linhas desnecessárias.

Figura 5 - Figura após a retirada da parte de água, deixando apenas uma linha.

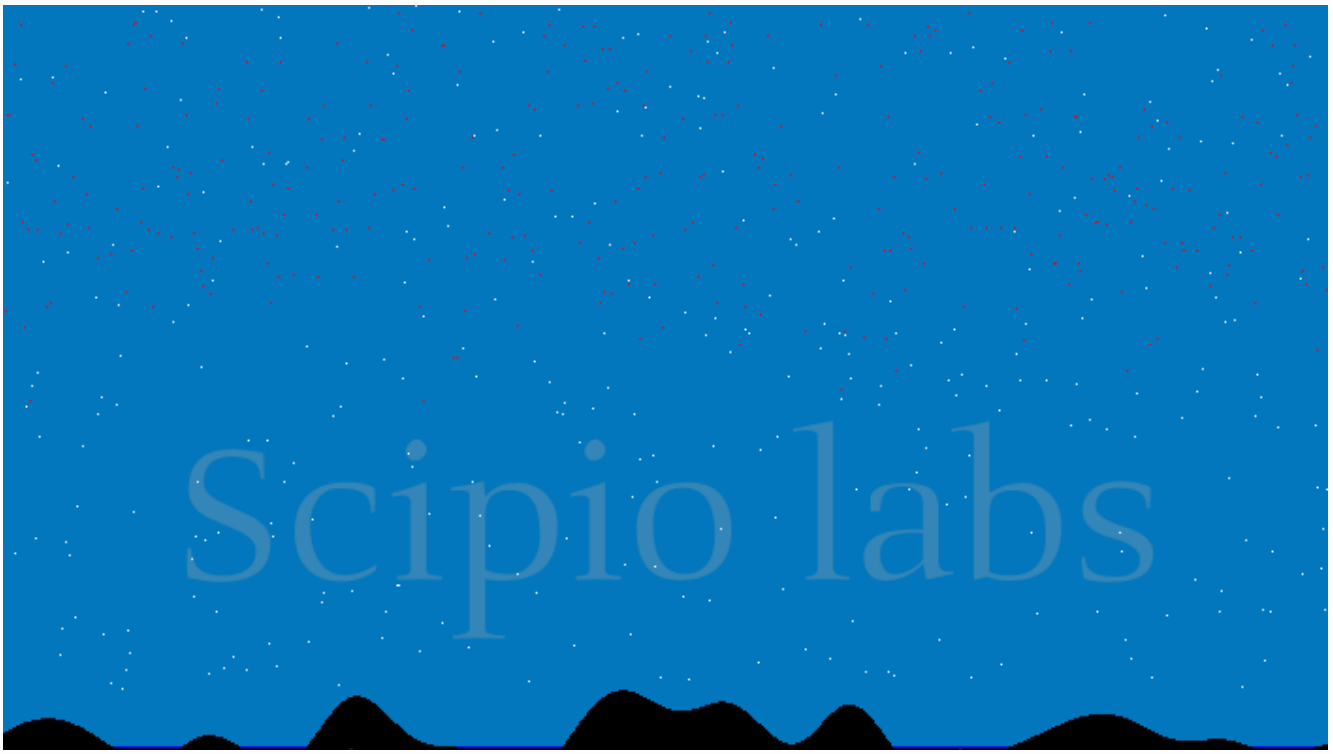

Figura 6 - Figura após a retirada das informações que apareciam depois da água.

Figura 7 - Figura após a retirada das informações entre a linha da última estrela e a linha de água.

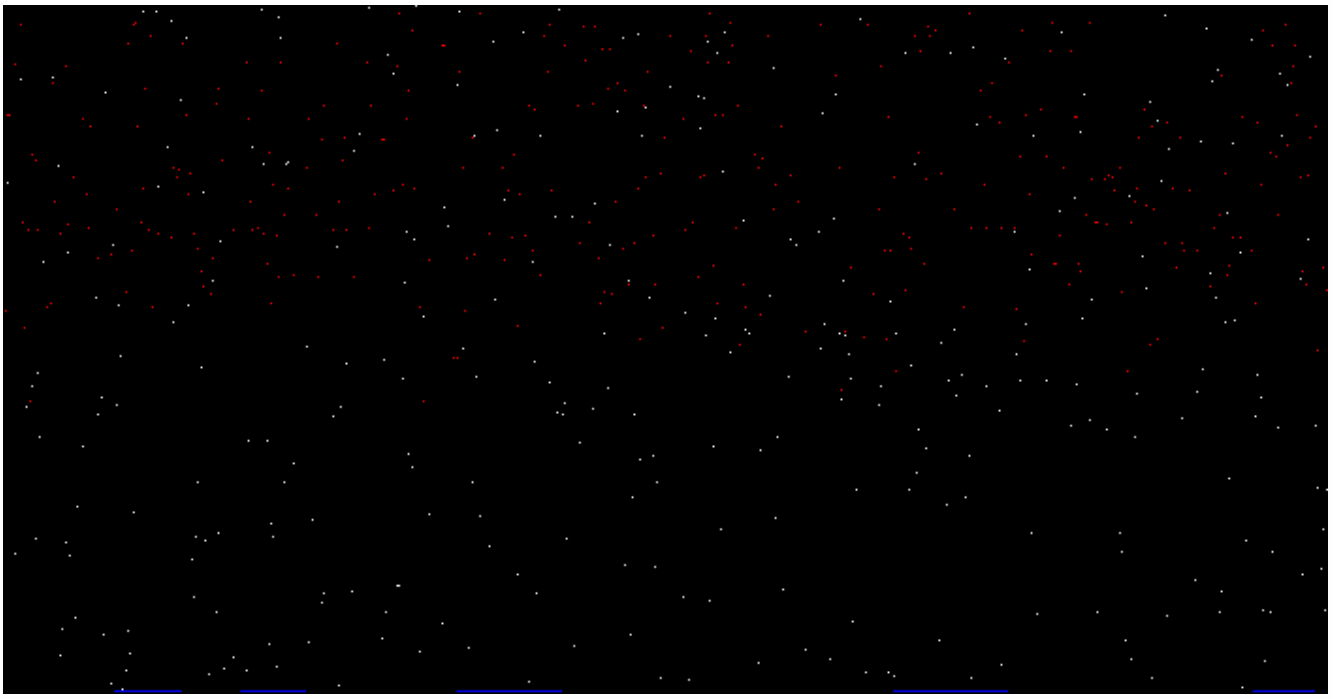

Figura 8 - Figura após executar a transformação de cor para apenas quatro.

Figura 9 - Figura após o “truncamento” da imagem.

Figura 10 - Figura após retirar as linhas pretas que não seriam usadas (uma foi deixada para análise visual).

8. **Resultados**

Número de estrelas	315
Número de meteoros	328
Número de meteoros que vão para água	105
(opcional); Frase escondida	Não encontrada

Tabela 2 - Resultados.

```
============================================================
=        Number of Stars           -        315           =
=        Number of Meteors         -        328           =
=  Meteors falling on the Water    -        105           =
=        Hidden Phrase             -      not found       =
============================================================
```

Figura 11 - Print da saída entregue pelo código, onde o “not found” foi apenas definido por mim, já que a decodificação não foi finalizada.

O caminho para encontrar a frase parece estar traçado, porém devido ao tempo não foi possível finalizar a parte computacional. Parece ser código morse, sua estrutura já está montada e só falta finalizar essa parte, essa é a melhoria para futuro, além de melhorar o uso computacional. Parece existir uma codificação além de um morse simples. Foi feito um código que retira as informações em morse simples de duas maneiras:

maneira 1: caso não tenha informação branca ou vermelha retorna espaço em branco, caso tenha em uma retorna dot (.) e caso tenha nas duas linhas retorna dash (-) saindo:

“.-. - -.....- --. . . .-. .-. --. ... - ... .- ... -. -... --..-.. .-. ... .- . -- .... --..-.- --. ..- --... - --.- . -- ---- --- -.- - -- . .. -- - --...- .- . .-. . --.. - -- -.-- --. - - - -- . . -- -... --. - - ... .- . .. -. .. - --. - . .-. ... .-. .. .- .. . -- ...- --. .- -- -..- --..-.. --. -.. - -- ... -- ---- --- .-. .-. ... -- . . --. .. --. .. .- .. - .-.. . .- --.- .-. . . ... . -- ..-- -- .- -- -.-. .-.. . .- ... -. ... --. . . .- .. -- . . -- -..- --. . . - .... -.. .- --.-... -- .- .-. .. -- ...- --. ..- --. - . --. . . .-. . .- .. . -....-- --- -. -- . - -- --. -- .-.. .- .... -- .-- -- .-”

maneira 2: caso não tenha informação branca ou vermelha retorna espaço em branco, caso tenha em uma retorna dash (-) e caso tenha nas duas linhas retorna dot (.) saindo:

“-.- . .-----. ..- - - -.- -.- ..- --- . --- -. --- .- .--- ..--.-- -.- --- -. - .. ---- ..--.-. ..- --. ..--- . ..-. - .. ..-. . .. - -- .. . ..---. -. - -.- - ..-- . .. .-.. ..- . . . .. - - .. .--- ..- . . --- -. - -- .- -- . ..- . - -.- --- -.- -- -. -- - .. ---. ..- -. .. .--. ..--.-- ..- .-- . .. --- .. .. -.- -.- --- .. - - ..- -- ..- -- -. -- . -.-- - -. ..-. -.- - - --- - .. --.. .. -. .. .-.- -.-- - -. --- .- --- ..- - - -. -- .. - - .. .-. --. ..- - - . ---- .-- -. ..-.--- .. -. -.- -- .. ---. ..- --. ..- . - ..- - - -.- - -. -- - .----.. ... .- .. - . .. ..- .. -.-- -. ---- .. -.. .. -.”

Ao fazer isso saem 181 informações e foi dito que seria apenas 177, uma análise é que o morse varia de uma a seis informações, temos algumas informações que possuem mais que 6, essas seriam retiradas e tentaria uma nova análise. Porém a implementação dessa parte não foi possível, mas a ideia do que seria feito é essa.

Após analisar mais detalhadamente seis informações estão com mais de 6 pontos morse fazendo com que a saída fosse 175 caracteres, o que não seria possível sair.