t1-2582708

AUTOR: Daniel Zaki Sommer

O primeiro desafio foi criar um loop que só encerrasse quando o valor de pegaProximoPixel fosse igual a 0xFFFFFFFF. Reparei que apenas colocar “pixel” dentro do while criava um loop infinito (provavelmente por causa da manipulação que ocorre com o pixel dentro do bloco de comandos). Portanto, criei uma variável “aux” para guardar o valor original da função até que o bloco de comandos seja reiniciado.

Dentro do loop, eu precisava criar um único byte formado por pedaços de outros bytes, sendo o tamanho de cada um desses pedaços delimitado por "n_bits". Utilizei um loop "for" que repetia o processo de obtenção dos próximos pixels e a manipulação dos seus bits de acordo com o valor de "n_bits". Esse processo era realizado até que fosse possível obter um byte completo. Durante a manipulação dos bits, eu precisei utilizar diversas operações de deslocamento e "or" para juntar os bits de cada pixel e formar o byte final que seria enviado para a função "enviaByteRBD".

Para a função de descompressão, realizei um processo bem semelhante, utilizando a mesma estrutura básica do while e do for da primeira função, com exceção da forma como os dados são tratados dentro de cada uma delas. Iniciei realizando uma cópia do byte obtido, para não perder os dados contidos no restante do byte quando houver os deslocamentos. Atrelei o primeiro deslocamento à posição do fragmento de bits, representado por uma variável “posicao”, assim, independente de onde se encontra o fragmento, ele não sofrerá perdas quando for deslocado para a esquerda do byte final.

No fim do processo, executei o programa para todas os arquivos que eu havia comprimido com a primeira função. O resultado foi positivo para n_bits valendo 2 e 4, com a imagem final sendo bem fiel à original, fora o fato de ela estar um pouco escurecida. Porém, para n_bits valendo 1, o resultado foi apenas uma imagem preta, para todos os testes:

n_bits = 4

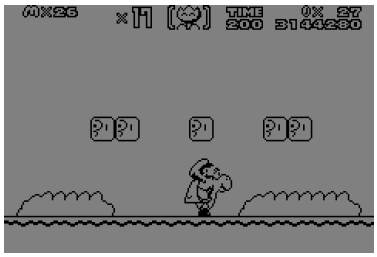
n_bits = 2

n_bits = 1

Para descobrir onde estava o erro, tentei descomprimir os arquivos originais enviados no .zip, para ver se o problema estava na compressão ou na descompressão. Para minha surpresa, as imagens saíram perfeitas para todos valores de n_bits, portanto concluí que o meu erro estava na compressão, que estava gerando imagens mais escuras. Para corrigir isso, estabeleci que quando o pixel for maior ou igual a 0x7f, ele deveria ser convertido para um valor maior, para haver um maior contraste na formação da imagem final e poder formar um desenho.

n_bits = 4

n_bits = 2

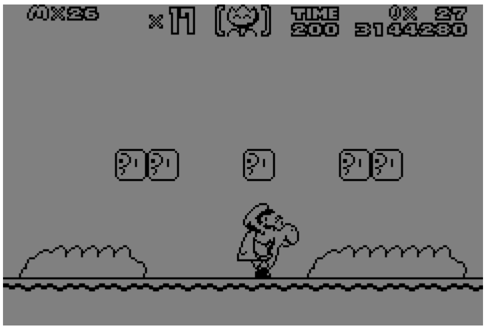
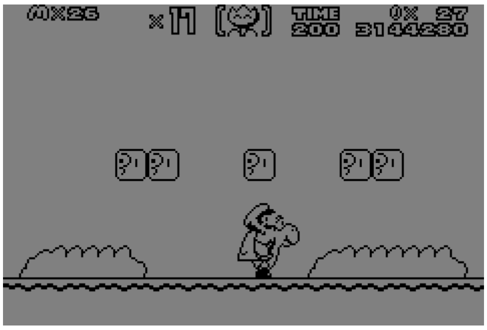


n_bits = 1

O resultado final foi bastante satisfatório para mim, ainda que possam haver algumas otimizações possíveis no código. Achei que a minha solução ficou bastante enxuta e que a implementação não deve enfrentar dificuldades. Procurei inúmeras formas de reduzir as perdas de bits que estava escurecendo as imagens, mas nenhuma das soluções encontradas seriam econômicas ou condizentes com o conteúdo até agora ensinado no curso. Acredito que talvez haja uma forma muito mais simples que eu não tenha enxergado. Mesmo assim, estou satisfeito :).

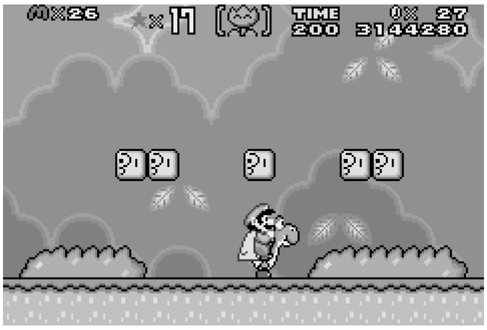
Descompressão dos arquivos .rbg gerados:

Descompressão dos arquivos .rbd originais:


TIME
200
3144280

TIME
200
3144280

TIME
200
3144280

TIME
200
3144280

TIME
200
3144280

TIME
200
3144280