Graphic Compression in Retro Video Games

Presented by- Ankit, Prabhakar, Yatharth

Introduction

Video Games have been around for a few decades now. Hardware limitations, and the lack of growth in the industry in general caused video game companies to run on a certain budget. In the 1980s, a new form of animation, called sprites was developed. These involved making multiple frames of a single object to cover the different animations. These sprites by themselves were not large, but could not be used until they were compressed due to memory complications. Therefore, several workarounds were deployed. Sprite compression involves decomposing a larger, meta-sprite into smaller sprites, and containing them within the graphics table. Decomposition of the sprites is a manual process. We aim to automate the decomposition to make it faster. This technique could also be used for several different types of data compression, by exploiting the repetitiveness of the data in general.

Sprite Based Video Games

Animation, in a simpler essence refers to a collection of moving imagery. In early in the 90's decade, and before, most of the video games developed followed a certain form of animation called sprite animation. This involved creating multiple frames of an object, as it performed a certain action. In computer graphics, a sprite is a two-dimensional bitmap that is integrated into a larger scene, most often in a 2D video game. Sprites were developed at Texas Instruments by Daniel Hillis. Originally sprites referred to independent objects that are composed together, by hardware, with other elements such as a background.

Examples of systems with hardware sprites include the Texas Instruments TI-99/4A, Atari 8-bit family, Commodore 64, Amiga, Nintendo Entertainment System, Sega Genesis, and many coin-operated arcade machines of the 1980s.

Sprite Based Video Games

Sprites are two-dimensional images or animations overlaid into a scene. They are the non-static elements within a 2D game, moving independently of the background. Often used to represent player-controlled characters, props, enemy units, etc., sprites can be composed of multiple tiles or smaller sprites. They can also be used for pseudo-3D sprite scaling like in Super Scaler, Mode 7, or in pre-rendered movement.

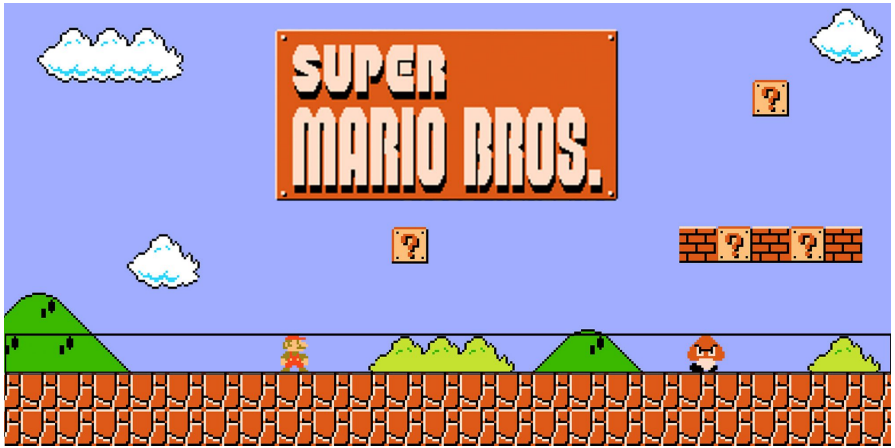
RPG Maker MV Character Sprite

Hardware Limitations

NES was a pinnacle at gaming consoles then, consisting of Ricoh 2A03, an 8-bit microprocessor containing a second source MOS Technology 6502 core, running at 1.79 MHz for the NTSC NES and 1.66 MHz for the PAL version, and 2kB of on-board RAM. The graphics are categorized into 2-background, and foreground. Background graphics consisted of data that did not move, hence stayed ‘static’. Foreground graphics referred to the moving parts of the game, which are called sprites.

Famicon by Nintendo Entertainment System

A SNES sprite can be 8x8, 16x16, 32x32 or 64x64. Sizes cannot be chosen freely. A game can have two of the predetermined size combinations: 8x8, 16x16 8x8, 32,32 8x8, 64x64 16x16, 32x32 16x16, 64x64 32x32, 64x64 This means that, once picked, the NES will consider every sprite in the game to be one of these sizes. Thus, if a game is using 8x8 and 16x16 sprites, a 32x36 character would actually be considered as 3 sprites. If the game was using 32x32 and 64x64 sprites, then the 32x36 character would be 1 sprite.



Super Mario is a game that everyone who has a prefix played.

Game Logic

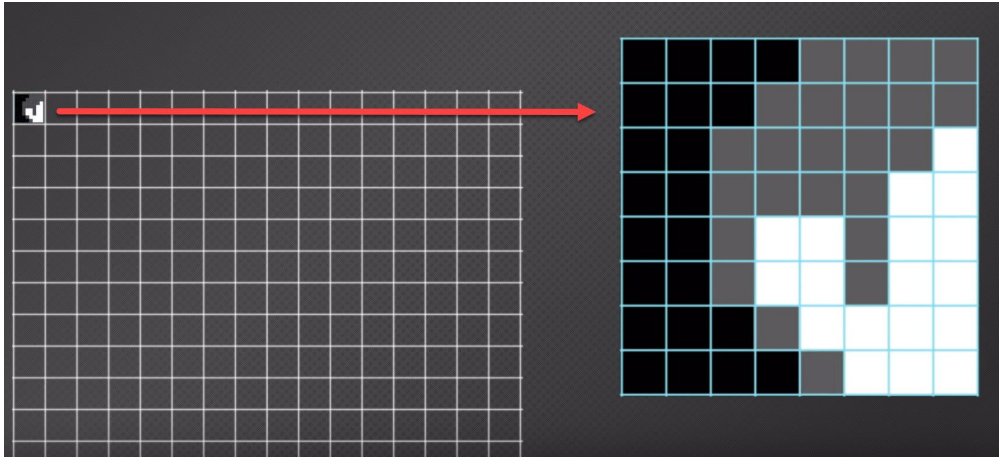
To make the game more interesting to play for more than five minutes, the developers set the following requirements:

1. This will be a platformer - a game where the main character needs to run and jump on the platforms, climb up and jump over obstacles.
2. The hero will be able to deftly move and shoot at enemies.
3. To be able to play with the company, they make multiplayer for four people.

On the one hand, Assembler code executes very quickly; on the other hand, it's working stupidly in it, transferring data from one processor cell to another. This is about how to cook sushi, working with individual rice.

The memory was allocated as follows:

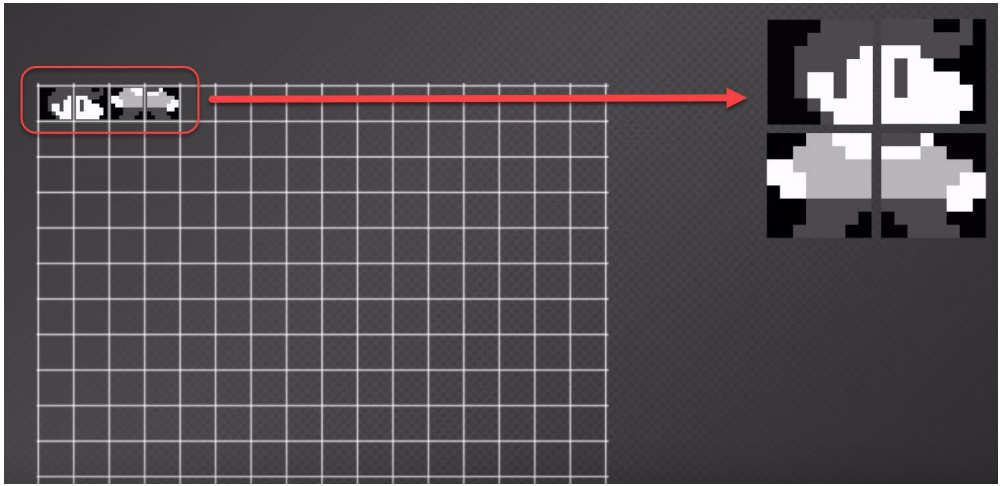
1. 8 kilobytes on the chart
2. 32 kilobytes for the game code itself & data storage.



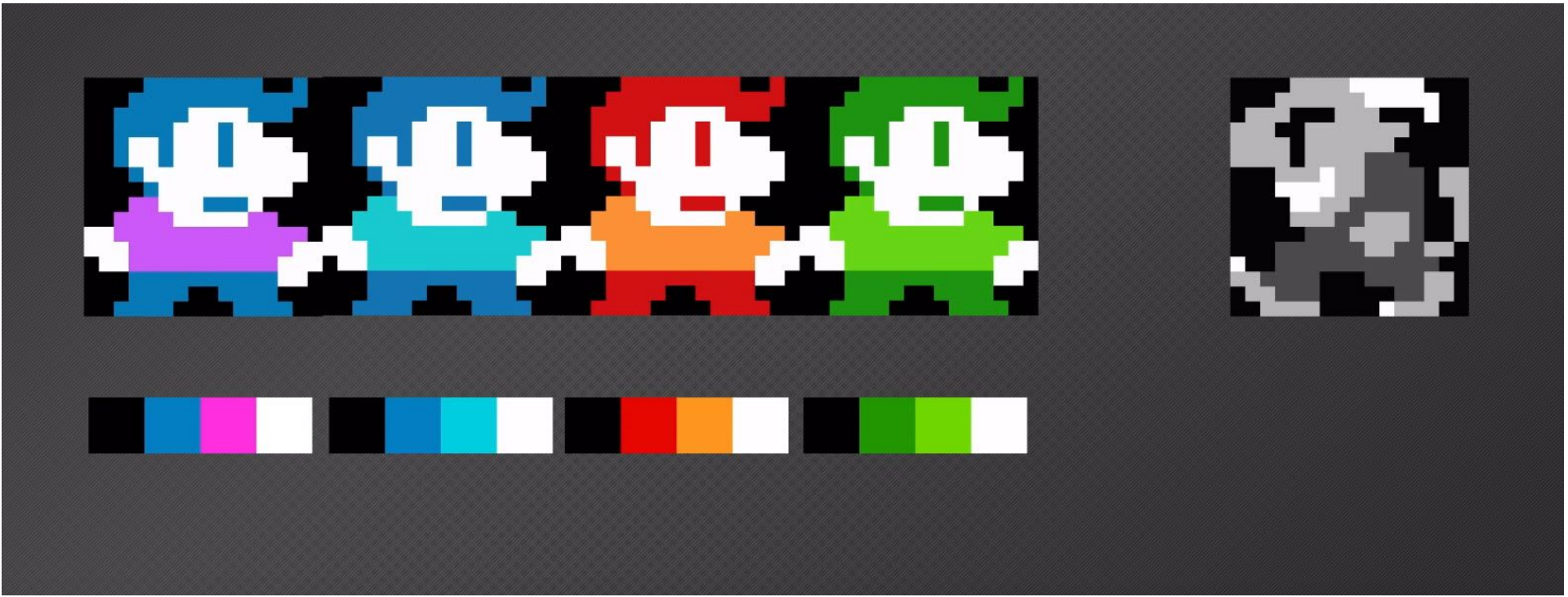
In each such square, you can draw something, but use only three colors.

Character

The game has two types of graphics: a static background and moving objects- players, opponents, bosses and shots. Everything that moves is called sprites. The developers divide the entire graphic memory into two parts - one for sprites, the second for the background:

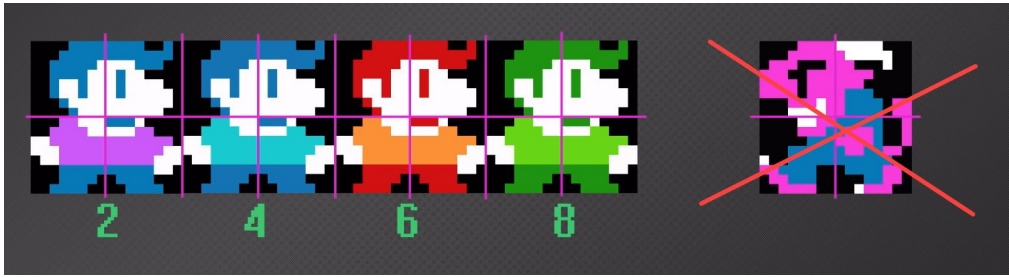


If you combine several squares into one, you get a metasprite.
In our case, a character

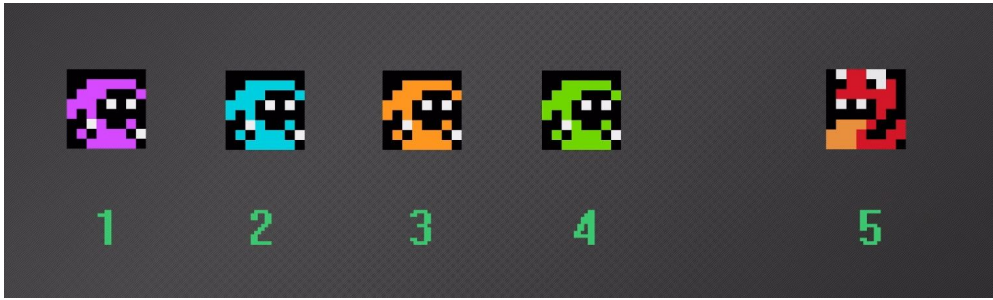


The prefix can use only 4 types of palette at a time, so we get 4 colored protagonists and an unpainted villain.

New limitation: there can be only 8 sprites on the screen at a time - there is not enough memory for more. Therefore, there is no room for the villain. You can go to the trick and show them quickly, quickly in turn, but then the picture will flicker and look worse.

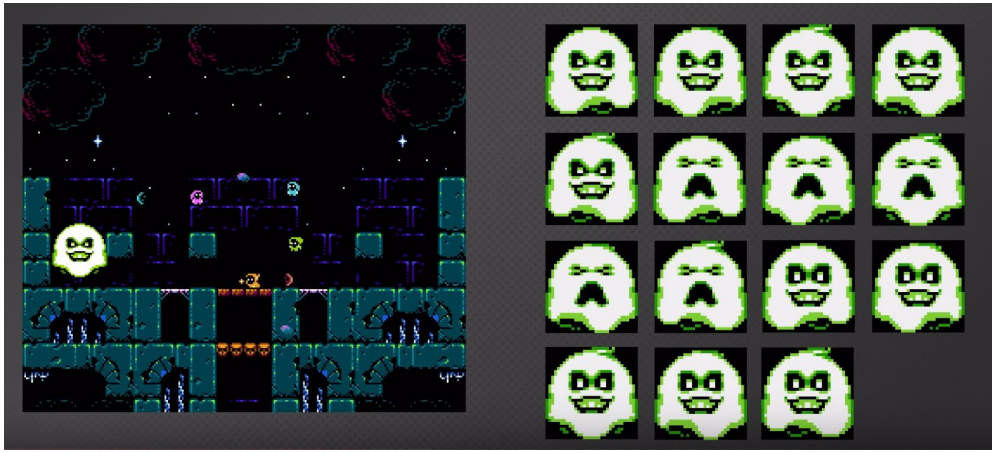


The developers have radically reduced the size of the heroes and the villain to one sprite. Now they look more arbitrary, but are placed on the screen.



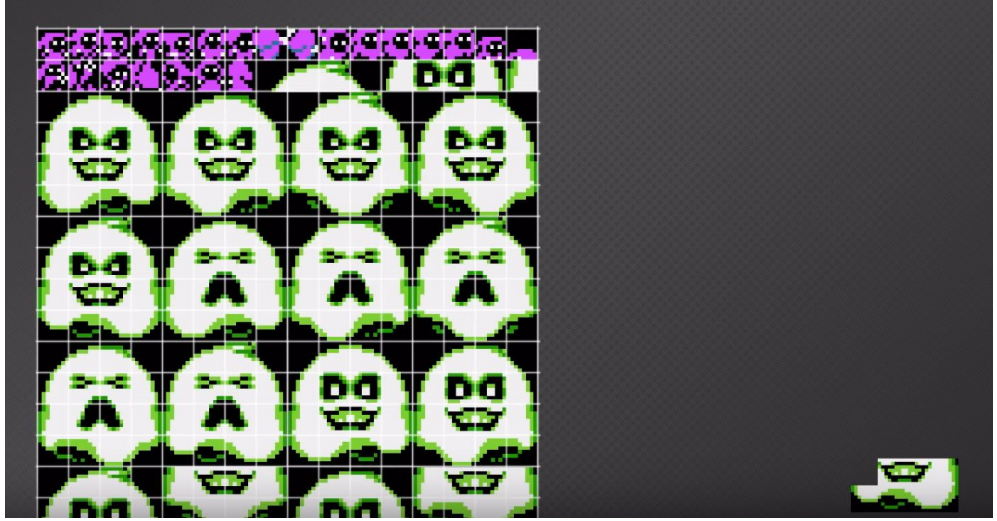
Graphical Workarounds

Most of the graphics ingame are taken by sprites, which refer to the ‘moving’ part of the graphics. The size of sprites depends, but a larger sprite is called metasprite. In essence, a single, larger sprite can be broken into multiple, smaller sprites. This is one of the most common techniques used to compress a certain sprite. It can be shown as the following.

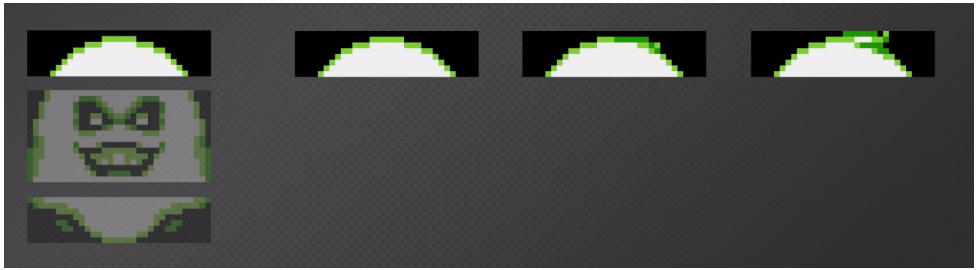


Big Boss and his animations options

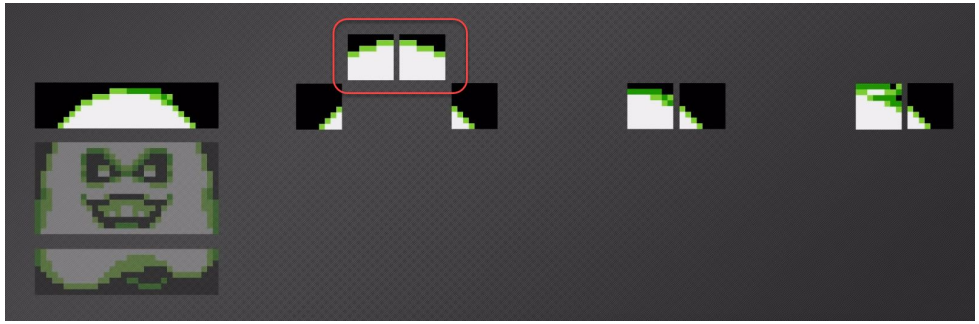
If we distribute all sprites in a table one to one, then we will quickly run out of space and one piece will not fit. Remember this picture as an example of non-optimized work with memory.



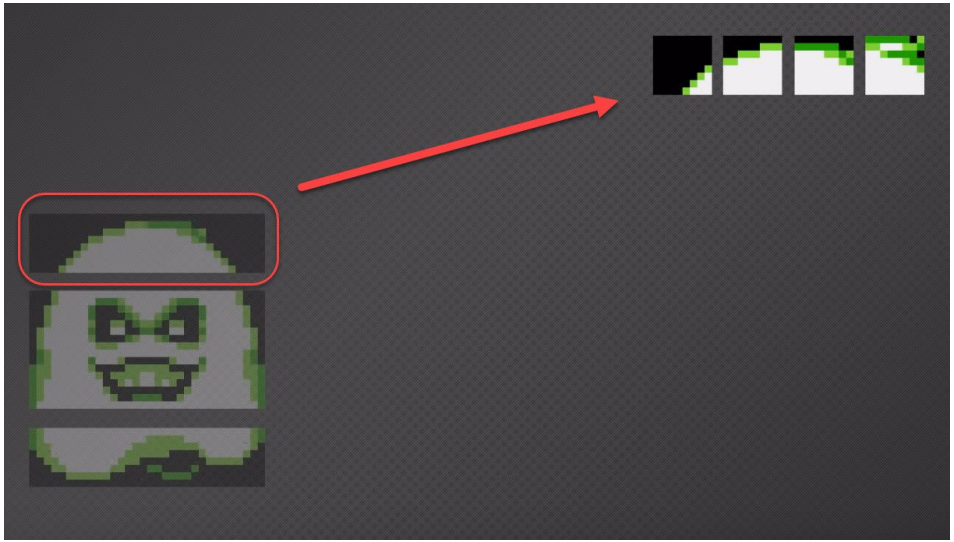
To begin with, the developers split the boss horizontally into three parts, and each is animated separately. It can be seen that the animation of the hairstyle consists of three pictures, each of which is slightly different from the rest.



And here it is clear that this is the same sprite, only in a mirror form. It is easy for a computer to draw it reflected, so you can also safely leave only one of them. With the last triangles in each picture - the same thing: these are the mirrored first sprites.

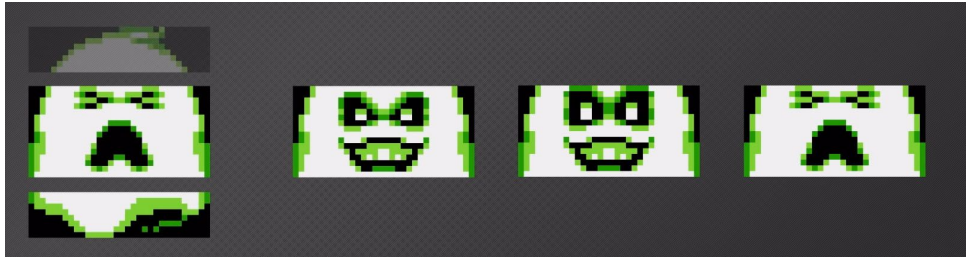


As a result, the entire upper part of the boss along with the animation fit in four sprites. This is the optimization: there were 16 sprites, it became 4.



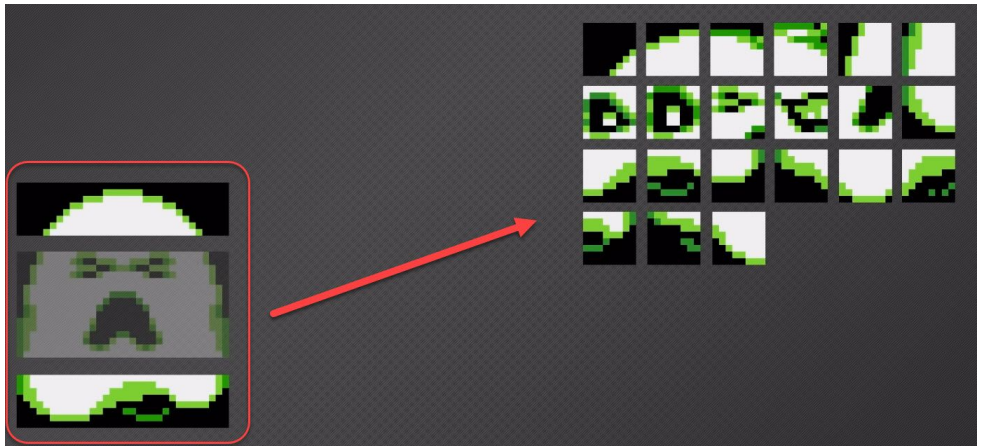
Do the same for the middle part.

Now it takes $3 \times 8 = 24$ sprites.

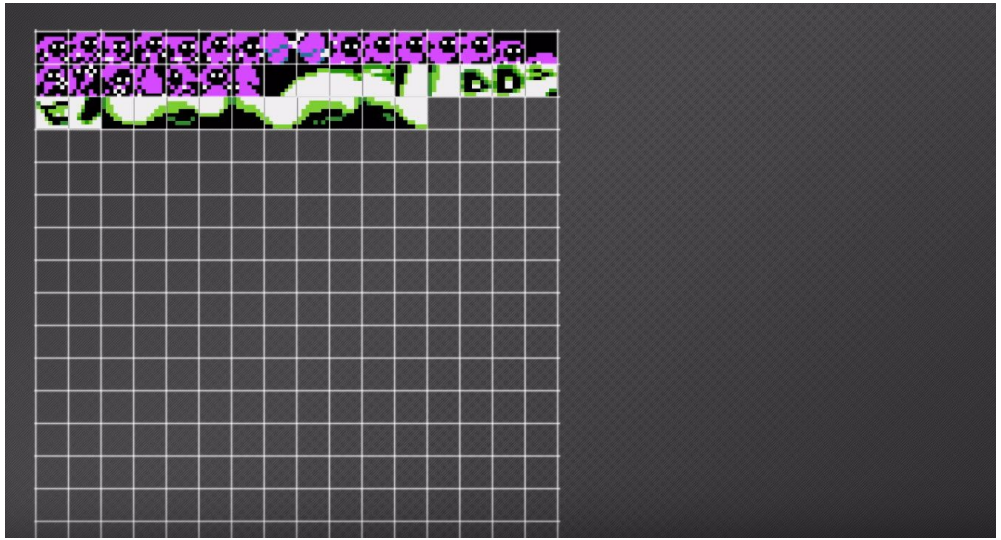


After full optimization, the boss takes only 21 sprites.

The final view of the boss is assembled from these pieces.



Another similar technique is Sprite Flickering. Sprite flickering on the NES would occur when more than 8 sprites were displayed on the same horizontal line. We kept the sprite count as low as we could, but as previously mentioned, we didn't sweat the exact numbers. Some of our objects produce a few more particles than an NES game would allow. Some games like Recca or Contra got around sprite limits by displaying certain sprites only every other frame (at 30fps instead of 60fps). On CRT monitors running low resolution interlaced video, objects would appear to be drawn every frame. In addition to this, NES particle art was often built with flickering in mind for effects like explosions.



Compare with original version before optimization

Automation

These are a few tricks that are used to reduce an image in an NES game. Most of the following is done manually, although it is not hard to code for the following. We have accomplished the following automation through the workflow as described below -

1. Split the images into an n x n matrix. (Note - n is a power of 2) It is preferable to split into an 8 x 8.
2. Compare the images one by one.
3. Flip the images and compare again.
4. Create a larger binary coded database to mark the sprites.

Cacodemon from DOOM

Upon comparing our results to the memory it would've taken were the sprites not decomposed, we reduced the entirety of sprites by 86%. The compression means that instead of taking 190 blocks, we compressed it into a mere 26 blocks. Such a compression is extremely useful to represent larger sprites.

The above image is a Cacodemon from the DOOM series. Although, the automation requires the user to enter the number, intuitively, we can split the following into 4 parts. Using the script on the following result was generated.

Splitted Cacodemon in 4 parts

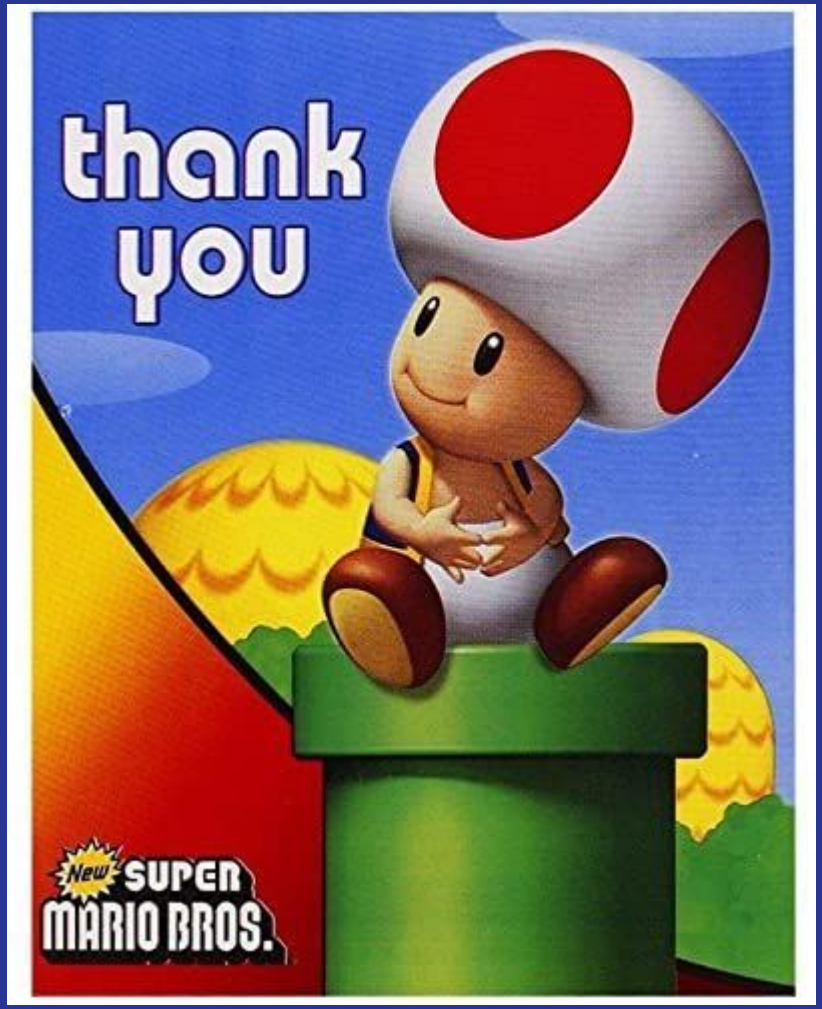
Using the sprite compression module, the entire sprite is reduced to 2 separate sprites, due to the fact that one side is a mirror of the other side.

Therefore, resulting in the compression of the following by 50%. Larger sprite sets, such as the Boo sprite set mentioned in the previous section can benefit more from the following compression algorithm. Even though not perfect, the following helps in reducing the amount of manual workload a sprite animator will go through in a certain optimization time.

Using Compression module, Cacodemon is reduced to 2 sprites

Conclusion

In this work we gave an introduction into console video game history as well as work related to digital preservation in general and emulation as a digital preservation strategy. We outlined the challenges for preserving console video games and discussed various different preservation strategies. We used the Planets preservation planning approach to evaluate digital preservation alternatives for an assumed library environment where console video games were to be archived as digital heritage. While only the major console video game systems of each era are covered in this work, most of the results are also applicable to other console systems.


thank
you
New SUPER
MARIO BROS.