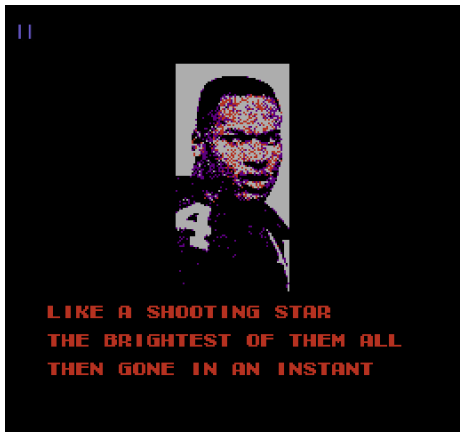
by "ovenmitt" – July 2021

Creating Tecmo Super Bowl player portraits

With a bit of creativity and some effort, it's possible to make some very cool native Tecmo graphics including player portraits. Having gone from absolute zero knowledge of NES or Tecmo-specific internals to customizing an entire game thanks to TecmoBowl.org, here's my guide to creating portraits. Obviously, you can use these techniques to create portraits of any person, it doesn't have to be a Tecmo player.



Prior to starting, you should consider exactly where you want to place the portrait. If it is just to create a PNG file, that's easy. If you want the portrait to actually appear in the game, you'll have to decide where. There aren't a ton of places, my favorite is replacing the NFL Player Association page in the intro credits. Although that graphic is small, it is comprised of 8 meta-tiles of 32 square pixels each, which means you have a 128x64 canvas on which to work.

This guide will have 2 sections: creating the graphic, then inserting it into a ROM cartridge file.

Section I – Creating a portrait

You will need:

- a screenshotting tool (PicPick is terrific: `https://picpick.app/en/`)
- a photo editing program, such as GIMP (free), Photoshop, Affinity Photo, etc.
- GraphicsGale (`https://graphicsgale.com/us/`)
- I-CHR (`https://kasumi.itch.io/ichr`)

Steps:

1. Find an image you want to use as your base image, and screenshot it. Remember that you'll be dealing with an extremely limited color space and small size – so crop tight on the player's face, don't use pics with lots of detail or additional objects, etc.
2. If your screenshot tool allows for some basic editing, like cropping, start with that. Your final image will be 64x128 (if you are planning to insert into the game in the title screen), so you want one axis to be exactly double the size of the other. It's also best if you can crop to some dimension of 16 pixels: 320x160 or 144x288, etc. You can also adjust brightness and contrast if the tool allows – most likely, you want to reduce brightness and increase contrast.
3. Eliminate the photo background. You can do this in PicPick or in your photo editor. Basically pick a brush and white-out everything except the player. Some photo editors may automate this process, but just manually outlining the player's face probably works best and isn't too difficult. You don't need to be perfectly precise, remember we will be reducing the size and color space.
4. Save the resulting image and open it in your photo editing program. Assuming you're using GIMP, generally speaking there are 3 adjustments:
    a. Add RGB Noise: Filters > Noise > RGB Noise. 0.20 to 0.30 works pretty well
    b. Resize the file to the final dimension of 64x128: Image > Scale Image

    c. Reduce colors via Posterize: Colors > Posterize. In most cases, use value of 3
5. If you choose to make additional adjustments to the colors, do it after resizing the image. Re-scaling will interpolate colors and add thousands of colors to even a small image, while our goal is to reduce the number of distinct colors.
6. Save the file as a PNG or BMP then open in GraphicsGale. Note there probably are ways to perform this next step in GIMP but Gale makes it very easy. Check the number of distinct colors in the image: Image > Count Colors Used. Now reduce the color count to 4: All Frames > Color Depth > 4bpp then manually change the number of colors from 16 to 4. Keep in mind the actual colors in the resulting image are not important – you will have to set the colors independently on the ROM file's palette. What is important is whether there is a sufficient amount of contrast and shape/interpretation of the original image.
7. Most likely you'll have to experiment and cycle through various iterations and adjustments to the colors, noise level, etc. You may also find that Gale doesn't really optimize to 4 colors very well – in other words, the resulting image only has a handful of pixels using the 4th color. In these cases, allow Gale to reduce to 5 or 6 colors, and then:
    a. Save file and open it back up in GIMP. Make sure the file is in RGB Mode: Image > Mode > RGB
    b. Manually set the 5th color pixels to match one of the first 4. Colors > Map > Color Exchange. Set the From color to what you want to change, then the To color to match one of the existing safe colors. Now you're back to a file with exactly 4 colors.
8. Save the 4-color file as a PNG. In File Explorer, you can simply drag the image file onto I-CHR.exe. This will create an output folder which includes an .nes ROM file that you can open directly in any NES emulator, among other files.

Section II – Inserting into a Tecmo ROM

You will need:

- a hex editor such as HxD (www.mh-nexus.de)
- YY-CHR (https://w.atwiki.jp/yychr/ or most NES dev web sites)
- an NES emulator (Mesen, FCEUX, etc)
- an expanded Tecmo Super Bowl ROM file, such as the TecmoBowl.org or SBlueman annual releases

Note it is extremely difficult to add new graphics to the original TSB ROM file, simply because there is no empty space to add them. Your only option would be to overwrite existing tiles and then very painstakingly re-assemble the tiles into an image. Even then, it's likely your new graphic will require more tiles than what it is meant to replace.

Steps:

1. Open the folder created by I-CHR. The 2 key files are anim.nes, which you can open immediately in an emulator and view your image, and chrblock.chr, which is the image definition. Unfortunately by default, anim.nes repeats the first metatile so that you won't have a clean background in most cases.
2. Open the chrblock.chr file in your hex editor. Assuming you created a 64x128 sized graphic, the file should be 0x800 bytes long. In other words, it should start at 0x000 and go to 0x7FF.
3. Also open your target ROM file in the hex editor. Copy the entire chrblock.chr file, then paste it in the ROM at location 0xA1810. This should be an empty block, but make sure you "paste write" (overwrite) and not "paste insert" which will increase the length of the target file.
4. After saving, you can also open the target ROM file in YY-CHR and see the tiles you just pasted in. You'll have to scroll down to 0xA1810, which is also known to the game as Tile Bank 86. Note that they will be out of order. If you need to make any last-minute edits (this would be highly unusual) you can do so here. You can also start to play around with the color palette, which is easy to change in YY-CHR.
5. The screen definition is found starting at 0x544E. Please see bruddog's excellent post here if you aren't familiar with screen definitions. Tecmo allows us to keep 2 banks of 128 tiles each in memory (each tile is 8x8 pixels), and a 64x128 image requires 128 tiles if there are no repeats. The first two bytes in the definition pick the 2 banks to load, keep 18 but change the second byte (0x544F) to 86 to point to Bank 86 that we created above. The 5th byte (at 0x5452) tells us how many metatiles (a metatile is a 4x4 square of individual tiles, so 32x32 pixels) tall the image will be, while the 6th byte tells us how many metatiles wide.

A vertical image (128 tall, 64 wide) will use 04 02 while a horizontal image will use 02 04. The 7th byte signals the location of the first metatile. 0B or 13 for vertical images, 0A or 12 or 1A for horizontal images.

6. Now, this is tricky, we need to update the tiles inside the metatiles. The location of the first metatile is 0x1584C. Every metatile starts with a palette indicator, which in our case is always 00. Then the next 16 bytes each correspond to a tile in our bank, in a 4-by-4 matrix. We're going to use anim.nes as our guide, but the problem is that it isn't arranged exactly the same way. anim.nes doesn't use metatiles so all 8 tiles in a row are consecutive. anim.nes also defines every tile on the screen, where we skip all the space outside of our graphic. And finally, anim.nes assumes the tiles are the first bank in memory, but they will actually be the second bank (you can see this by opening the CHR Viewer in your emulator).
    a. If possible, open anim.nes in another hex editor that you can set to 32 bytes wide. I like to use an online editor such as `https://hexed.it/`, then go to Settings > Number of Bytes per Row > 32
    b. You'll likely need to insert 4 bytes at the top or right after the NES header row. You'll notice anim.nes contains many rows of repeating tiles, these are the spaces outside of our graphic
    c. The graphic probably begins at 0xE0 (may be different if you built a different shape or size). You can copy the first 4 bytes (0xE0 to 0xE3) to 0x1584D in the ROM. However instead of copying the next 4 bytes from the source file, you need to go down a row to 0x100 and copy those 4 bytes. Repeat 4 rows, so you've copies 16 tiles in total, or one full metafile. This should fill up 0x1584D to 0x1585C. Metatile 2 is sourced from 0xE4 to 0xE7 and then down 4 rows again. Keep alternating in 4-by-4 blocks until you've copied all 8 metafiles. Note that you can completely ignore the last 24 bytes on each row of the source file.

1	2	3	4	18	19	20	21
5	6	7	8	22	23	24	25
9	10	11	12	26	27	28	29
13	14	15	16	30	31	32	33
35	36	37	38	52	53	54	55
39	40	41	42	56	57	58	59
43	44	45	46	60	61	62	63
47	48	49	50	64	65	66	67
69	70	71	72	86	87	88	89
73	74	75	76	90	91	92	93
77	78	79	80	94	95	96	97
81	82	83	84	98	99	100	101
103	104	105	106	120	121	122	123
107	108	109	110	124	125	126	127
111	112	113	114	128	129	130	131
115	116	117	118	132	133	134	135

    d. Now we need to adjust for the second bank in memory. This means that for every tile you just copied, you'll need to add 80 (in hex) to each. So 04 → 84, 24 → A4, 4B → CB, etc.
    e. Suggestion: do this process just a few bytes at a time. Load your target ROM file in an emulator (Mesen is great for this step), and constantly check the results. You can run the emulator at a high speed for quicker feedback. You should be able to tell if you're on the right track just by viewing the first metatile – if the resulting corner of the overall image doesn't look right, try again. This is definitely the trickiest part of the process to get right. Each time you load new tiles, save the ROM, Reload in Mesen, and re-check.
    f. In total, you'll be editing from 0x1584C to 0x158D3. The next byte is 55, this is not part of the screen so don't change it.
7. Update the palette. Your 4 colors are defined at 0x1A0F0. In reality it's just 3 colors – you can change the first byte but that won't change the color from black. But you may experiment with changes to the next 3 bytes to see how the image looks. If you aren't familiar with NES palettes, there's a good explanation here by jstout. What is important to remember is that the colors themselves are not perfectly fixed or defined, so different emulators will have slightly different looks. In other words, deeper reds, or reds that are closer to pink, etc. With under 64 colors in total, don't expect to always find the exact color shade you'd like.
8. Another benefit of updating this specific screen is that you can add text. Starting at 0xC127, you can add you own caption or message. Line breaks are inserted with byte value B1, and since your graphic will cover a good part of the screen, you may want to start with a few line breaks before any text. If you absolutely need to use punctuation, open the screen in the CHR Viewer, pick the appropriate tile from the CHR Viewer and use that byte value – for example a comma is byte value 5C.

Summary

Here are the changes we made to the starting ROM file:

- 0x544E: Changed the screen definition to use a new tile bank, and perhaps to change the size and shape of the graphic (from horizontal to vertical), and the starting metatile location.
- 0xA1810: Added a new bank of tiles
- 0x1A0F0: Changed the color palette for this screen

- 0x1584C: Edited the metatile definitions in order to assemble the new tile bank into the proper arrangement
- 0xC120: Edited the text displayed on the screen

Advanced Topics

Note that I haven't fully tried the following, but it seems possible:

The initial pointer to the screen definition is the 2 bytes starting at 0x5034 (3E B4). You could change this to point to a new, clean space such as 00 BD, pointing to 0x5D10. By creating an entirely new screen definition, we are no longer constrained by having space for just 8 metatiles. So you could potentially create a double-sized image (128x128) by using both bank 86 (what we already use) and also 88 (or 84), then defining a 4x4 grid and pointing to 16 metatiles (although finding space to add 8 new metatile definitions would be tricky). This would eliminate the ability to add text, because those characters require bank 18 to be loaded, we would be overwriting that in the first byte of the screen definition. You could even envision a hybrid bank that copies the letter tiles and uses the rest for custom graphic tiles.

We use just 4 colors but technically, NES can load 13 colors at a time. We set the palette pointer to 00 at the start of every metatile definition, meaning we only use the first 4 bytes, or colors. It is certainly possible to use different palette pointers to use more colors – how palette pointers work is an entirely separate topic, but the major restriction is that while the screen can use 13 colors, any individual tile (an 8x8 block of pixels) can only use 4 colors. Actually it is more complicated than that – each block of 2x2 tiles, or 16x16 pixels, can only use 4 distinct colors. Therefore, the primary difficulty here would be in creating a source image in which the additional colors are isolated to just certain sections of the image. It might be possible to create a small area of an image which would benefit from a different highlight color – maybe a helmet logo, for example – but an entire image where each section used a different set of 4 colors would seem exceptionally difficult to create.