CREATING GRAPHICS FOR ATARI 8-BIT COMPUTERS

Graph2Font

A Graphician's Guide

mOdmate

Revision history

22 Dec 2025	v 1.0	First public release

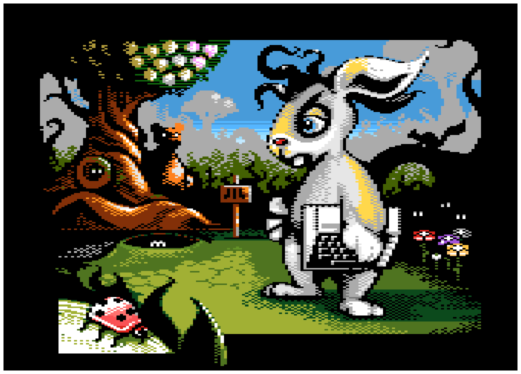
Fonts used: Roboto, Sofia Sans Semi Condensed

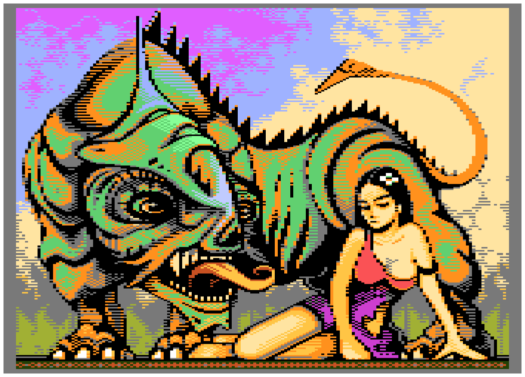
Cover image by Ripek (full image on p.53)

Interior images by Carrion, mOdmate, Odyn1ec, Piesiu, Powrooz, Ripek, Rocky, Tiger

Preface

It must have been around the year of 2010 when I accidentally discovered some pixel graphics for the Atari 8-bit that had just been released at a Polish demoscene party. Because I grew up with an Atari XE in the 1980s, I thought I would have an idea how Atari 8-bit graphics could look like. But in the light of these competition entries, I was so wrong! How could these masterpieces be displayed on an Atari XL/XE, with so many colors? Unbelievable! I was curious and started to look for more modern pixel graphics for that system—I must have missed a lot over the last decades! Soon enough I found more masterpieces by Powrooz (or Ooz) and Piesiu…


Follow the Easter Rabbit by Powrooz (2008)

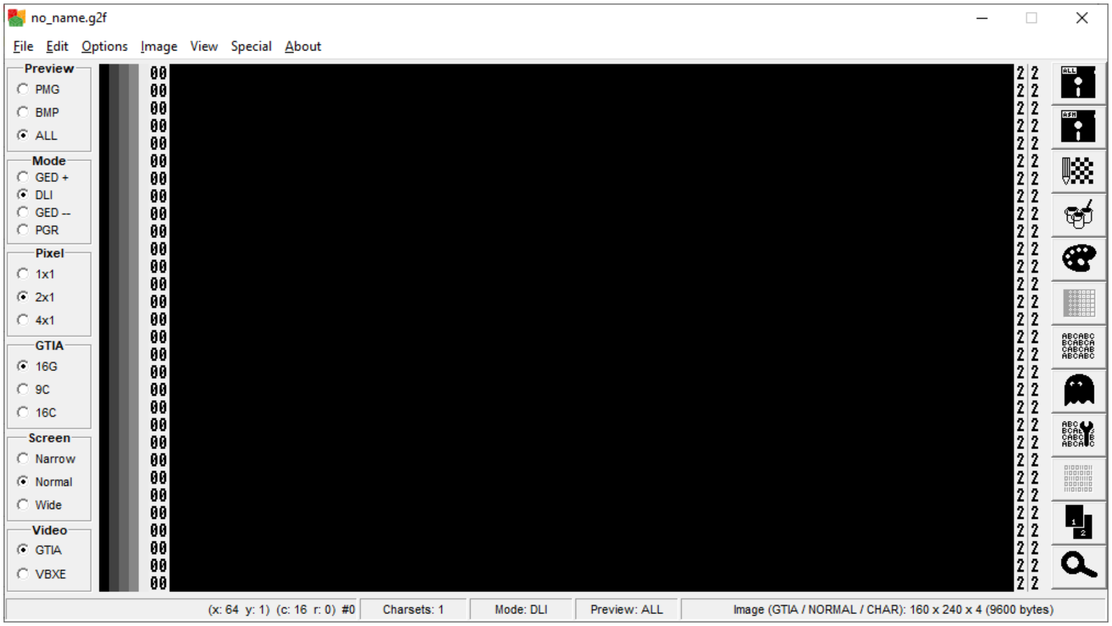

Pray by Powrooz (2011)

Dinner by Piesiu (2010)

Slumber by Piesiu (2010)

My fingers were itching to give it a try! Enthusiastically, I asked a friend from the ST demoscene if he knew which software was used to create these pictures. He murmured words like "*Graph2Font*", "*a Polish graphics program*", "*painting with the font*" and "*DLI*". It wasn't rocket science, but almost! My enthusiasm was a little dampened. Creating pixel art on the Atari ST was so easy: setting colors, entering zoom mode, pushing pixels and in a flash the picture was done (this is possibly a highly simplified recollection).

So, I visited the website of Graph2Font (http://g2f.atari8.info/), downloaded the latest version, installed and started it…



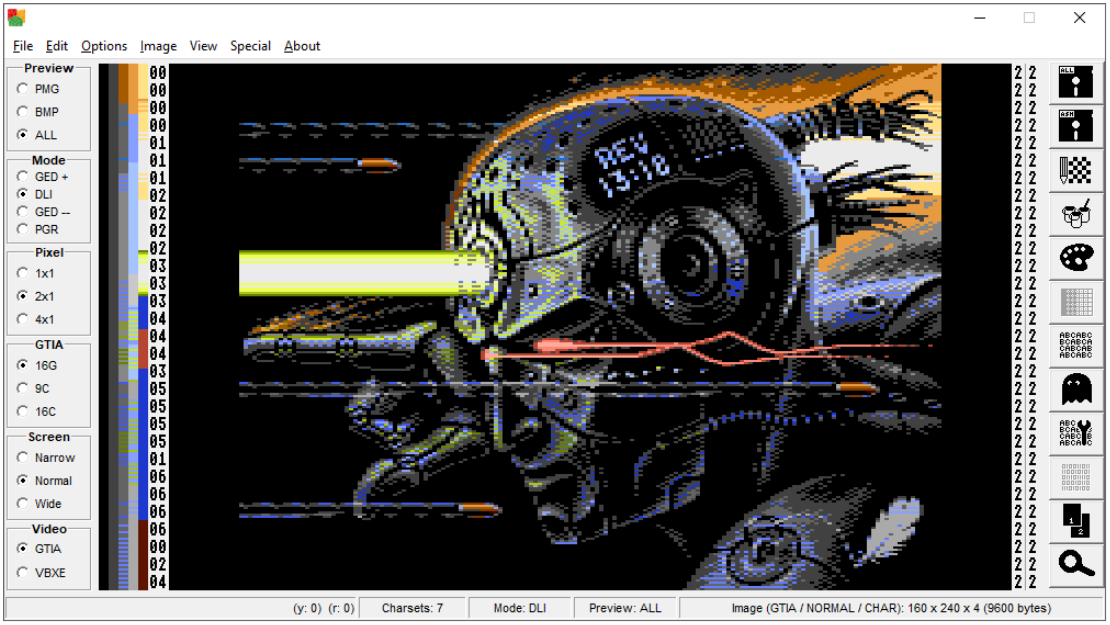
Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow [✓ Normal] Wide
Video: [✓ GTIA] VBXE

(x: 64 y: 1) (c: 16 r: 0) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

The Graphical User Interface of Graph2Font v4.0.2.8.

… only to quit after a few clicks: too many options that looked like Greek to me, too many mysterious abbreviations, and the handling of the painting tools needed some getting used to. In short: I was pretty disillusioned. Moreover, didn't I already need to paint several graphics for an ST demo in the making? And by the way, how does that fullscreen programming technique on the ST work? And so, my plans to produce pixel graphics for the Atari 8-bit computers disappeared in the drawer…

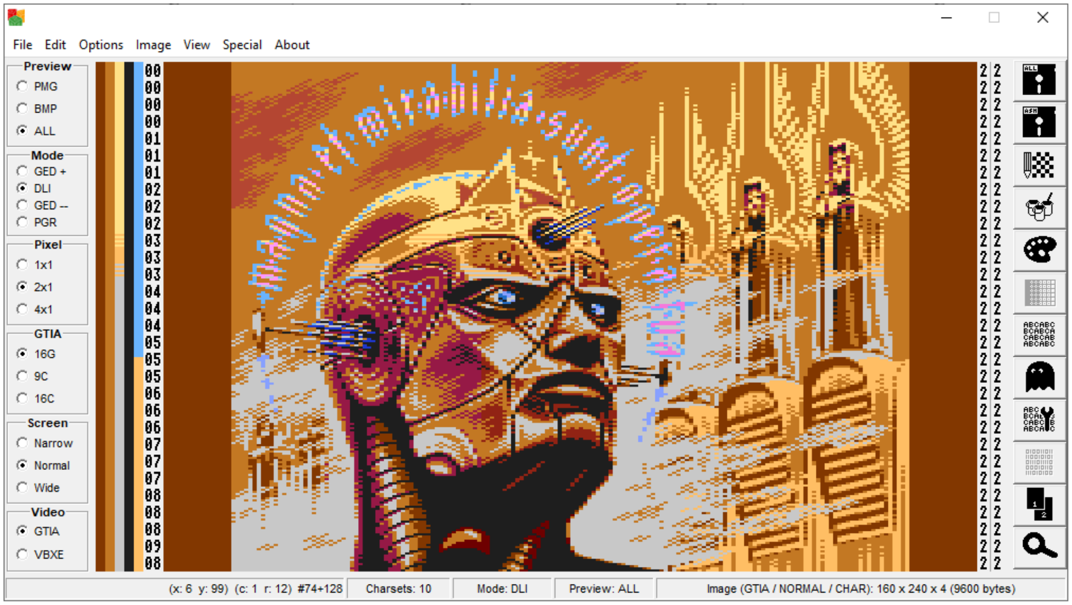
Fast forward to November 2016 in Gdansk, Poland. At a grocery store near the Silly Venture demoscene party, I met the famous and friendly Odyn1ec, the winner of the Atari 8-bit graphics competition two years ago and just about to win this competition again with another awesome picture.



Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow [✓ Normal] Wide
Video: [✓ GTIA] VBXE

(y: 0) (r: 0) | Charsets: 7 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

I Came Here To Destroy Your World by Odyn1ec, using 27 colors (2014).



Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow [✓ Normal] Wide
Video: [✓ GTIA] VBXE

```
(x: 6  y: 99)  (c: 1  r: 12)  #74+128 | Charsets: 10 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)
```

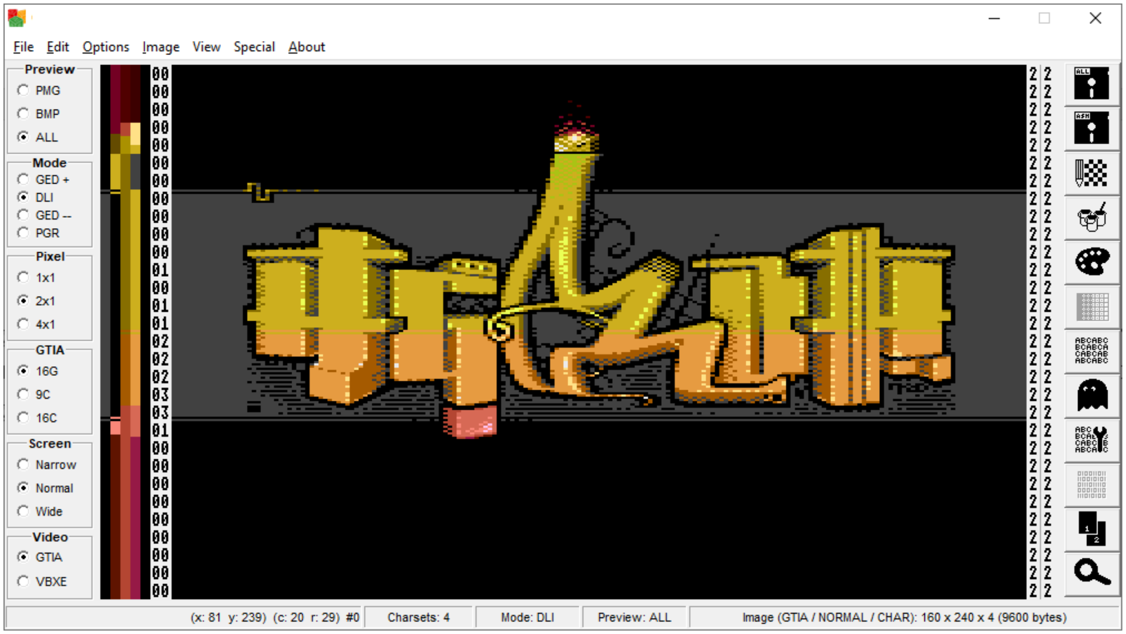
Crown Of Eternal Glory by Odyn1ec, using 16 colors (2016)

Odyn1ec immediately offered to show me how to work with Graph2Font (often shortened as “G2F”)—but unfortunately, we lost sight of each other and the secrets of that software remained unrevealed to me.

Months passed until I found enough time besides all my projects for the Atari ST to make a second attempt. I was so motivated! I wouldn’t be put off once more, not this time! Graph2Font, here I come! Fortunately, when I got back in touch with Odyn1ec, he was a patient teacher who explained me most of the secrets and answered dozens of my questions. But despite being motivated, real life intervened, I changed my job, immediately had a lot of other things on my plate, and forgot about Graph2Font.

Until Christmas 2020 came. Finally, there was plenty of spare time for me. Clicking through the folders on my hard drive, I got to a folder named “G2F“. A little hesitant, I opened the program once more, checked the graphics user interface and said to myself: “*Let’s give it a try, maybe a logo as a starter could work*“. Some weeks and a lot of e-mails sent to Odyn1ec later, my first logo for the Atari XL/XE was done. Isn’t it ironic that it was not a logo for Odyn1ec's group Lamers, but for their main rival Agenda? Simply because I immediately had an idea for such a logo.

Man, I was proud that I finally understood—or should I write "defeated"?—Graph2Font and found a way to materialise my ideas for my beloved Atari 8-bit computer!



Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow [✓ Normal] Wide
Video: [✓ GTIA] VBXE

(x: 81 y: 239) (c: 20 r: 29) #0 | Charsets: 4 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

All beginnings are difficult: my logo for Agenda using 21 colors (2021)

Why another handbook?

Simply because there is none available yet. Even if the first versions of Graph2Font (G2F) were released back in 2003, no easily accessible teaching material describing the workflow from a pixel artist's perspective is available. You can find an old video course about G2F on YouTube, but it only covers the conversion of pictures from the Commodore 64 and explains the options of a particularly old version of G2F that looks quite different today. There are some manuals on the website of G2F—but they are either written in the Polish language, presented from a coder's perspective, or just not precise enough. Hence, I feel there's a gap in the current offer and this book aims to fill that gap—because the Atari XL/XE computers deserve more love from pixel artists from all platforms <3

If you are a pixel artist who knows how to push pixels on other systems and who is interested in doing graphics for the Atari 8-bit systems, this guide is for you. Due to the scope of this book, I won't explain every single option and tool that G2F offers. I won't describe the use of the drawing tools that are commonly found in other pixel graphics editors, such as Draw, Line, Box, etc. Instead, I will focus on the tools that a seasoned pixel artist needs for creating Atari 8-bit graphics specifically.

Rick's Revenge by Carrion (2017)

About the author

I've been interested in computer graphics since the 1980s. Growing up in former East Germany, my first computer was an Atari 800 XE with a tape drive, which had a huge impact on me (not only in terms of patience!). In 1991, I switched to the Atari ST where I started learning pixel painting with programs like *PAD*, *Deluxe Paint ST* and *CrackArt*. Heavily attracted by the demoscene because of its colorful productions and kind people, I provided many pixel graphics to demos on the Atari ST over the last 30 years under the nickname m.O.d. or mOdmate (as seen on https://demozoo.org/sceners/2104/) but the 8-bit Atari computers still hold a very special place in my heart.

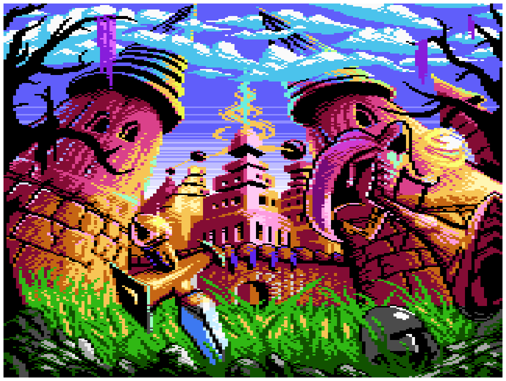
Note from Exocet:

Our friend mOdmate sadly passed away in 2025. When he realized he would not be able to finish writing this book, he asked if I would be willing to take it to completion. I accepted without hesitation, especially because his crash course at Sommarhack 2024 had been instrumental in my understanding of Graph2Font and the enigmatic world of Atari 8-bit graphics. mOdmate completed about 90% of the work on this guide, and it's clear he poured his heart into every page. I hope you find the content as compelling as I did, and that it helps many artists navigate the subtleties of creating pixel art pieces for the Atari 8-bit machines.

Rest in peace, Torsten. I imagine you would have been really happy to see this book finally out.

Getting started

To create nice graphics with G2F, you only need two things besides some good ideas: First, you should download the latest version of G2F (v4.0.3.1 as I'm writing these lines) from the official website on https://g2f.atari8.info/. You also need a pixel graphic program that can display pixels in a 2×1 aspect ratio, like Grafx2. That's because the biggest part of your work will be done in the pixel graphics tool of your choice and not in G2F (at least when you are working on a bigger project). Of course you can use G2F for that, too. But I don't see much sense in going the hard way here—it will be hard enough, I promise! :-D



The Lost Sorcerer by Tiger (2020).

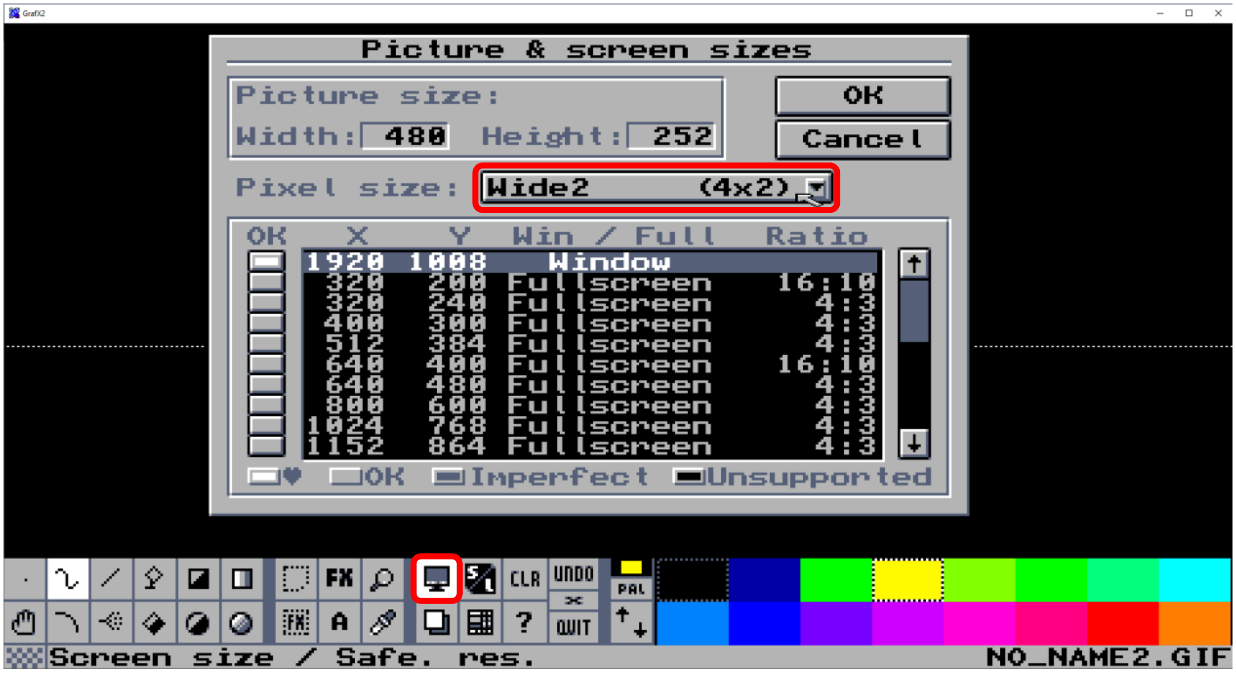
Preparation of Grafx2

As Grafx2 is my favourite pixel graphics package, I am going to explain how you need to set it up for painting Atari 8-bit pixel graphics. You should take care of three things:

1. Displaying the correct **pixel aspect ratio** (2×1).
2. Using the correct Atari 8-bit **color palette**.
3. Using the **image resolution** of the Atari 8-bit screen.

Displaying the correct pixel aspect ratio

To set the correct pixel size, click on the monitor icon on the toolbar of Grafx2 with the left mouse button. A panel called *Picture & Screen Sizes* will appear, where you should choose either *Wide (2×1)* or *Wide2 (4×2)* as the *Pixel Size*. Both settings are appropriate, but I suggest using *Wide2 (4×2)*, because it makes things a bit easier to see for smaller images like the one we are about to work on.



Picture & screen sizes

Picture size:

Width: 480 Height: 252

OK

Cancel

Pixel size: Wide2 (4x2)

OK	X	Y	Win / Full	Ratio
▭	1920	1008	Window	
▭	320	200	Fullscreen	16:10
▭	320	240	Fullscreen	4:3
▭	400	300	Fullscreen	4:3
▭	512	384	Fullscreen	4:3
▭	640	400	Fullscreen	16:10
▭	640	480	Fullscreen	4:3
▭	800	600	Fullscreen	4:3
▭	1024	768	Fullscreen	4:3
▭	1152	864	Fullscreen	4:3

▭♥ ▭OK ▭Imperfect ▬Unsupported

Screen size / Safe. res. NO_NAME2.GIF

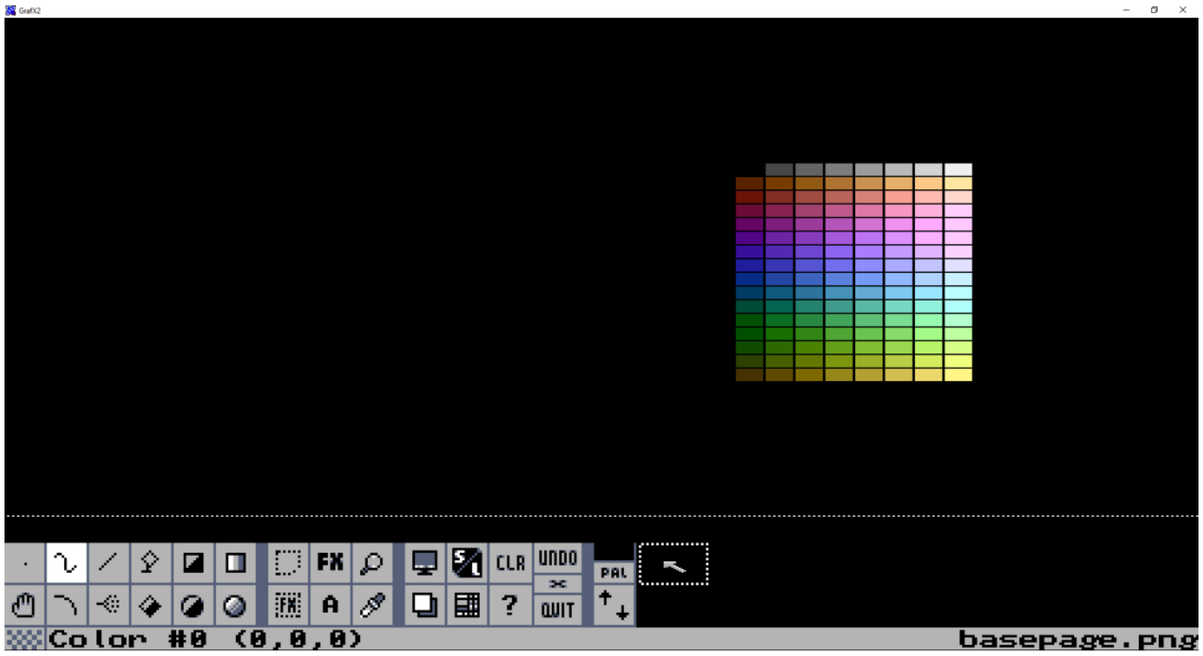
Setting the size of pixels to 2x1 in Grafx2.

Setting up the Atari 8-bit color palette

Secondly, we need to know which colors we are allowed to use on our picture. The color palette of the Atari XL/XE computer series is a bit…special. In the graphic mode we are going to use, the color palette consists of 128 entries, but among other peculiarities, it doesn't contain a good red tone, for instance. Many interpretations of this color palette exist, which differ sometimes a lot, sometimes only in nuances.

If you are curious about the different color variants, you can check them in the Altirra emulator in *View > Adjust Colors* and then select the different presets, or directly in Graph2Font with *View > Palette > External Palette > Browse*.

To play it safe, I simply made a screenshot of the whole color palette of G2F and prepared a color grid in Grafx2 that I can access at any time while working on an image.



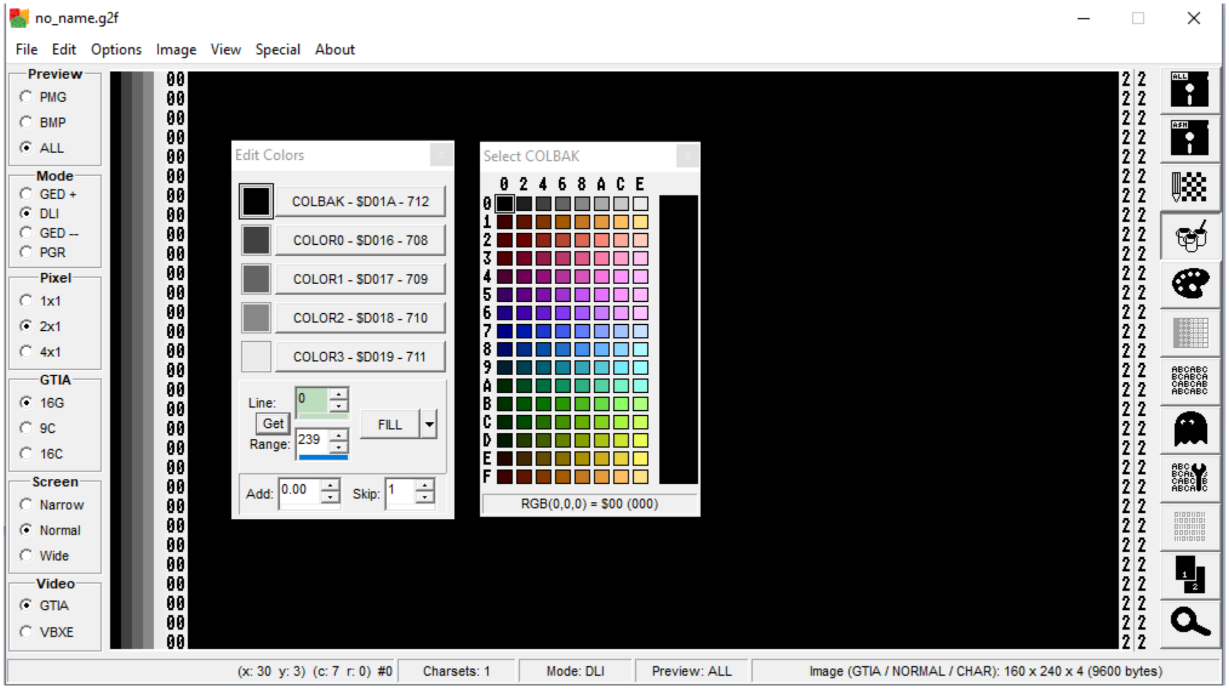
Preparing the basepage in Grafx2 with a full color palette of the Atari XL.

In Graph2Font, to get an overview of all the available colors, you can select *Edit Colors* in the *Edit* menu (shortcut: **Alt+C**) and then click any of the five colored boxes on the left side of the *Edit Colors* window. Another window will then open, showing the full palette.

Alt+C

Edit the colors of certain lines of the image

Sounds complicated? Don't panic—my basepage (PNG image file) comes along with this handbook and is available here.



Preview: PMG, BMP, [✓ ALL]
Mode: GED +, [✓ DLI], GED --, PGR
Pixel: 1x1, [✓ 2x1], 4x1
GTIA: [✓ 16G], 9C, 16C
Screen: Narrow, [✓ Normal], Wide
Video: [✓ GTIA], VBXE

COLBAK - $D01A - 712
COLOR0 - $D016 - 708
COLOR1 - $D017 - 709
COLOR2 - $D018 - 710
COLOR3 - $D019 - 711
Line: 0
Range: 239
Add: 0.00
Skip: 1

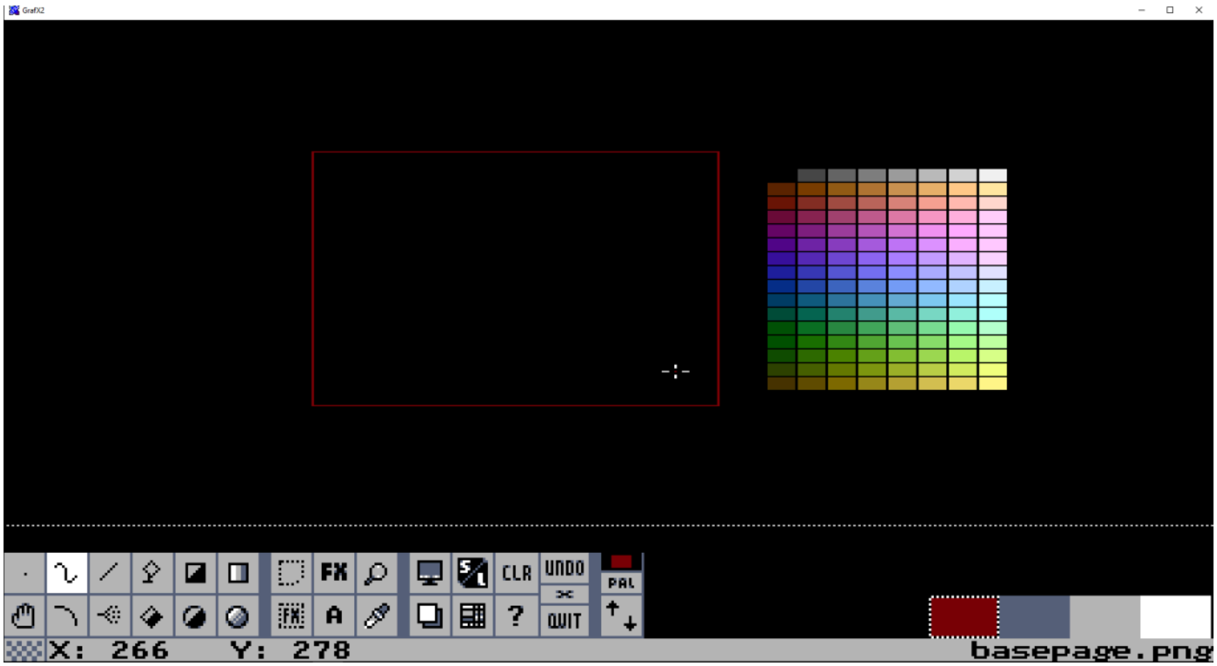
RGB(0,0,0) = $00 (000)

(x: 30 y: 3) (c: 7 r: 0) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

Showing of the full color palette of the Atari XL.

Image resolution

The third and last step of our preparations is related to the resolution in which our final picture will be displayed. In practice, it's nothing more than "emulating" the screen of the Atari XL/XE. The ideal screen resolution is 160 pixels in horizontal direction and 240 in the vertical direction (160×240).

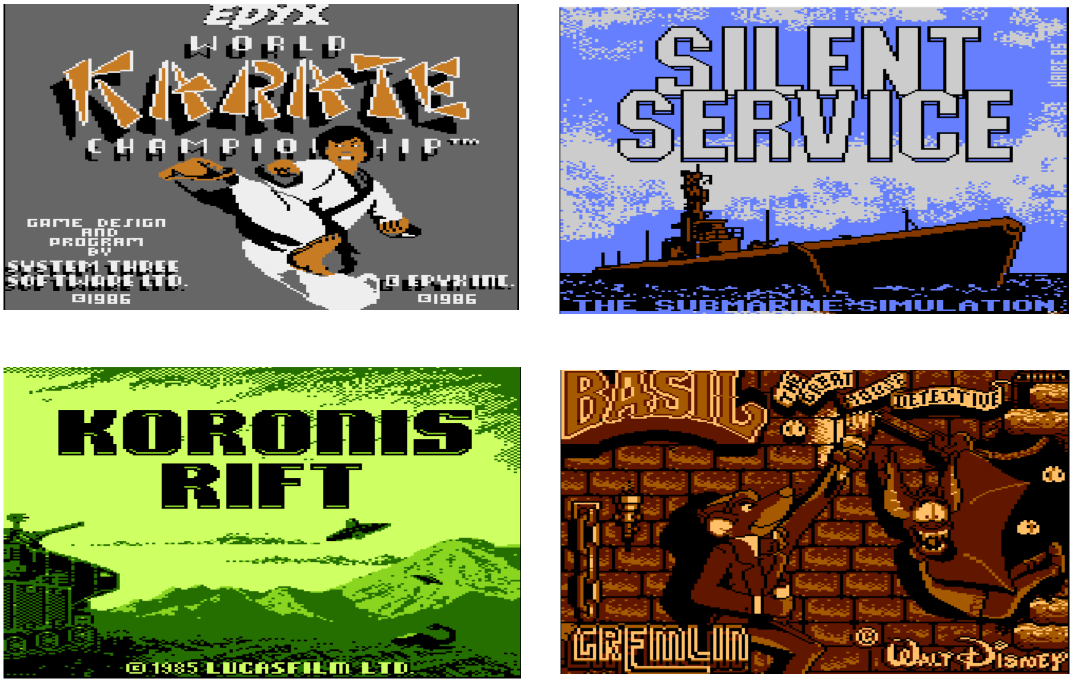


Final basepage in Grafx2 with a 160x240 px frame and a full color palette of the Atari XL/XE.

Some fundamentals about the graphical capabilities of the Atari XL/XE

The 8-bit Ataris have several graphic modes with different screen resolutions and color ranges. I won't go through all of them in this guide as we will focus on a single graphic mode.

The graphic mode we will use can only display four different colors simultaneously on the same screen line. You might be familiar with this graphic mode if you've seen title screens of games from the 8-bit Atari's golden age in the 1980s...



EPYX
WORLD KARATE CHAMPIONSHIP™
GAME DESIGN AND PROGRAM BY SYSTEM THREE SOFTWARE LTD. ©1986
© EPYX INC. ©1986
SILENT SERVICE
THE SUBMARINE SIMULATION
KORONIS RIFT
© 1985 LUCASFILM LTD.
BASIL THE GREAT MOUSE DETECTIVE
GREMLIN
© WALT DISNEY

Title screens of Atari 8 bit games from the 1980s using four colors.

Talking about this graphic mode, I'd like to reiterate two things:

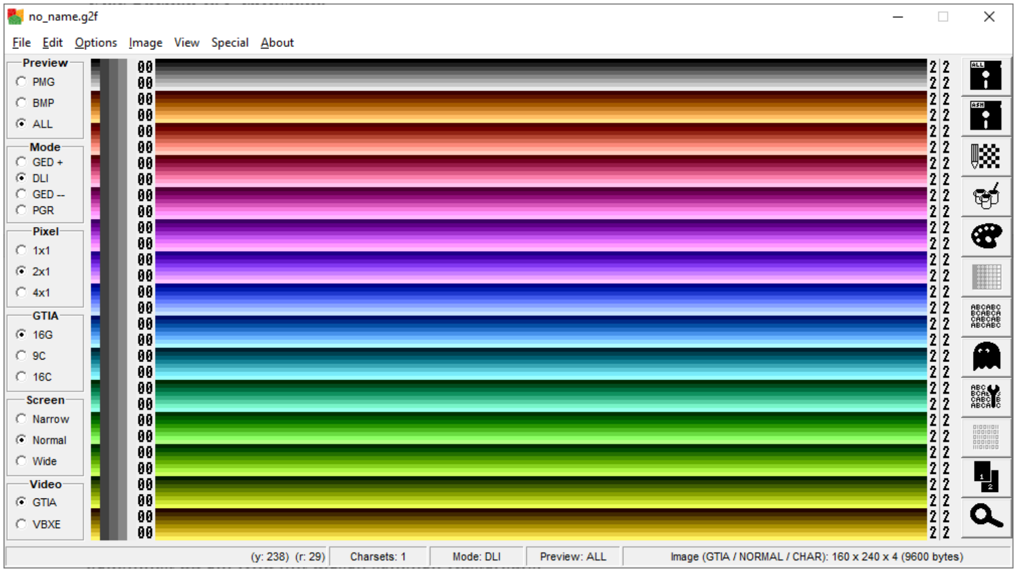
1. You can display a maximum of four different colors per line.
2. I am only referring to the "base" *Bitmap* graphics—but there are other graphical elements that can have different colors. I will explain that in a minute.

Four colors per line

If you already created graphics on the Atari ST, you may know that the color palette can potentially be changed at the end of each screen line (or even within) by a certain coding technique. Thus, you can display more colors on the screen than the system specs may indicate (16 per screen for the Atari ST).

The Atari 8-bit computers can perform such color changes as well by programming the Display List Interrupt (DLI). But no worries, you don't have to code that by yourself—this job is done by Graph2Font for us! It's up to you if you change one, two or up to four colors per line or if you don't change a color at all. The only important thing for you to know is that you are able to change your palette at the end of each line, so the next line appears with new colors.

For instance, you can change the background color (called *COLBAK* in Graph2Font) on every second line and get a picture with rainbow raster bars (it's really done with only a few mouse clicks, as I will explain later):



Preview: PMG BMP [✓ ALL]

Mode: GED + [✓ DLI] GED -- PGR

Pixel: 1x1 [✓ 2x1] 4x1

GTIA: [✓ 16G] 9C 16C

Screen: Narrow [✓ Normal] Wide

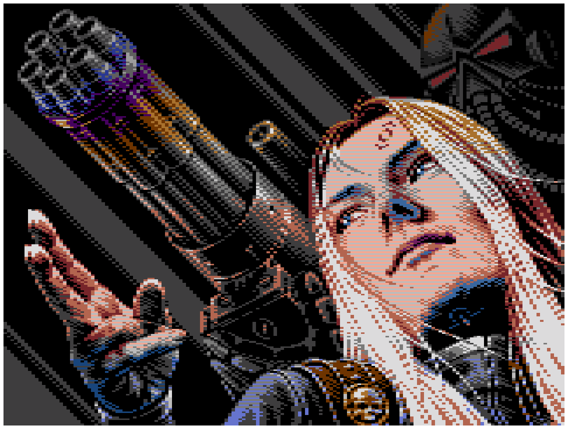
Video: [✓ GTIA] VBXE

(y: 238) (r: 29) | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

Changing the background color every second line results in nice raster bars!

Bitmap graphics vs *Player* and *Missile* graphics

Now we are getting to a fairly Atari-specific aspect, because the *Bitmap* graphics layer in 160×240 pixels (as used here) is not the only graphical element the Atari XL/XE can display. Probably with game developers and numerous gamers in mind, Atari included hardware sprites in their 8-bit machines called *Players* and *Missiles*. In each picture a total of four *Players* and four *Missiles* can be displayed, and they are adjustable both in size and in form. They use a single color that can be any entry from the palette, and they can overlay *Bitmap* graphics or be overlayed by them. These characteristics make *Players* and *Missiles* a perfect additional element to your graphics!



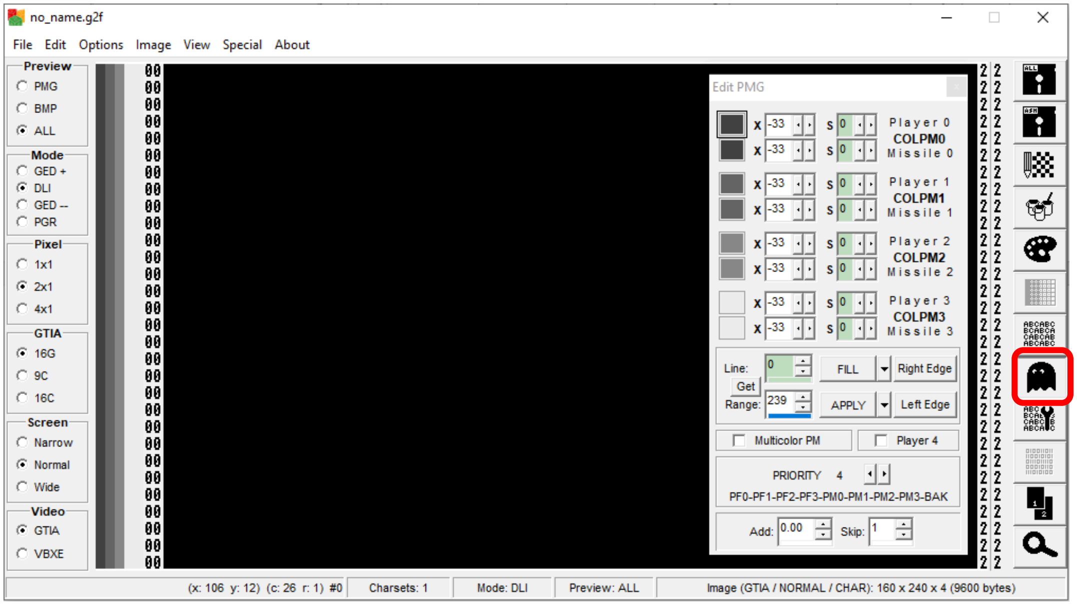
Sister of Mercy by Odyn1ec (2018).

However, there is one special feature when painting on the *Player/Missile* graphics layer which might feel a bit odd compared to "freeform" pixel painting: *Players/Missiles* have first to be positioned on the screen using the *Edit PMG* function, which can be found in the *Edit* menu. Alternatively, you can press **Alt+P** or click on the ghost icon in the toolbar on the right side of the main window. After the positioning is done, you can shape your P*layer/Missile objects* in a second step using the zoom.

Alt+P

Shows the *Player/Missile* options

The *Edit PMG* panel might look a bit confusing when you look at it for the first time—but let's take a closer look at it. At this point of our guide, we should only be interested in the fact that there are four pairs of *Players/Missiles* called *COLPM0*, *COLPM1*, *COLPM2* and *COLPM3*, each with their own colors, horizontal positions, and sizes. We will learn about the details later.



Preview: [✓ ALL]; Mode: [✓ DLI]; Pixel: [✓ 2x1]; GTIA: [✓ 16G]; Screen: [✓ Normal]; Video: [✓ GTIA]

x -33, s 0 (each Player and Missile 0–3)

Line: 0; Range: 239

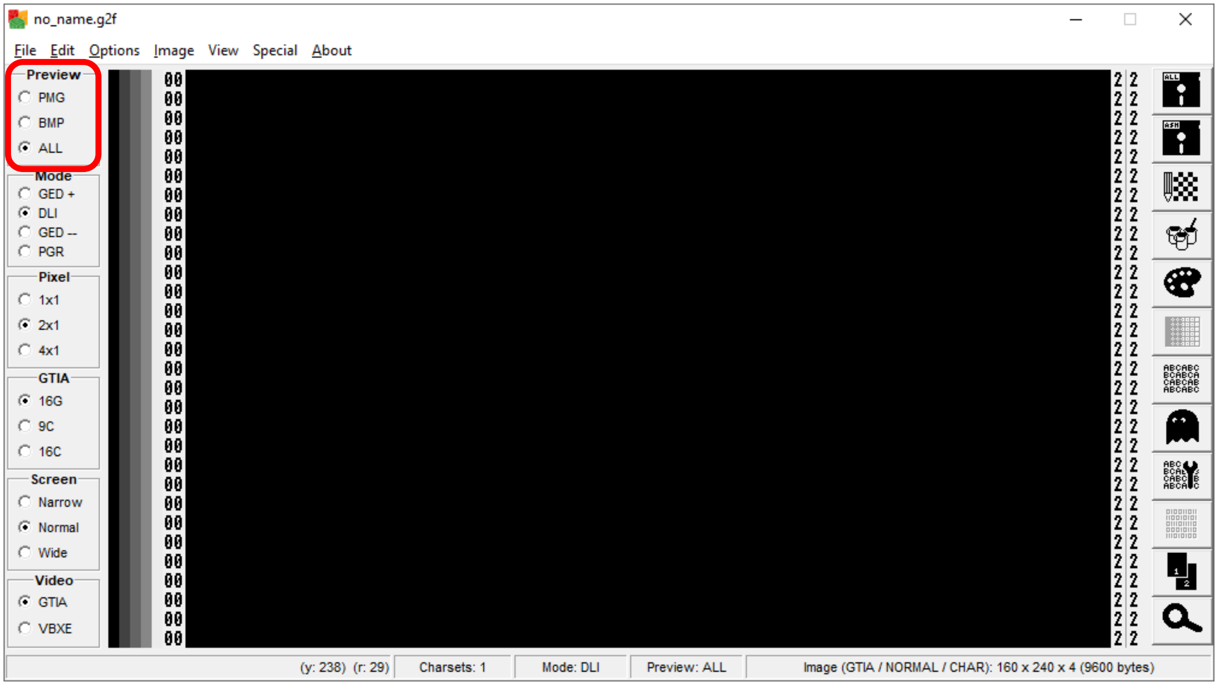
PRIORITY 4
PF0-PF1-PF2-PF3-PM0-PM1-PM2-PM3-BAK

Add: 0.00; Skip: 1

(x: 106 y: 12) (c: 26 r: 1) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

The *Player/Missile Graphics* menu.

To display the graphic layers (*Bitmap* and *Player/Missile*) independently, you can choose your preferred view in the *Preview* option in the menu on the left side of the main window. *PMG* only displays the *Player/Missile* graphics layer, *BMP* only displays the *Bitmap* graphics layer and *ALL* displays...guess what?



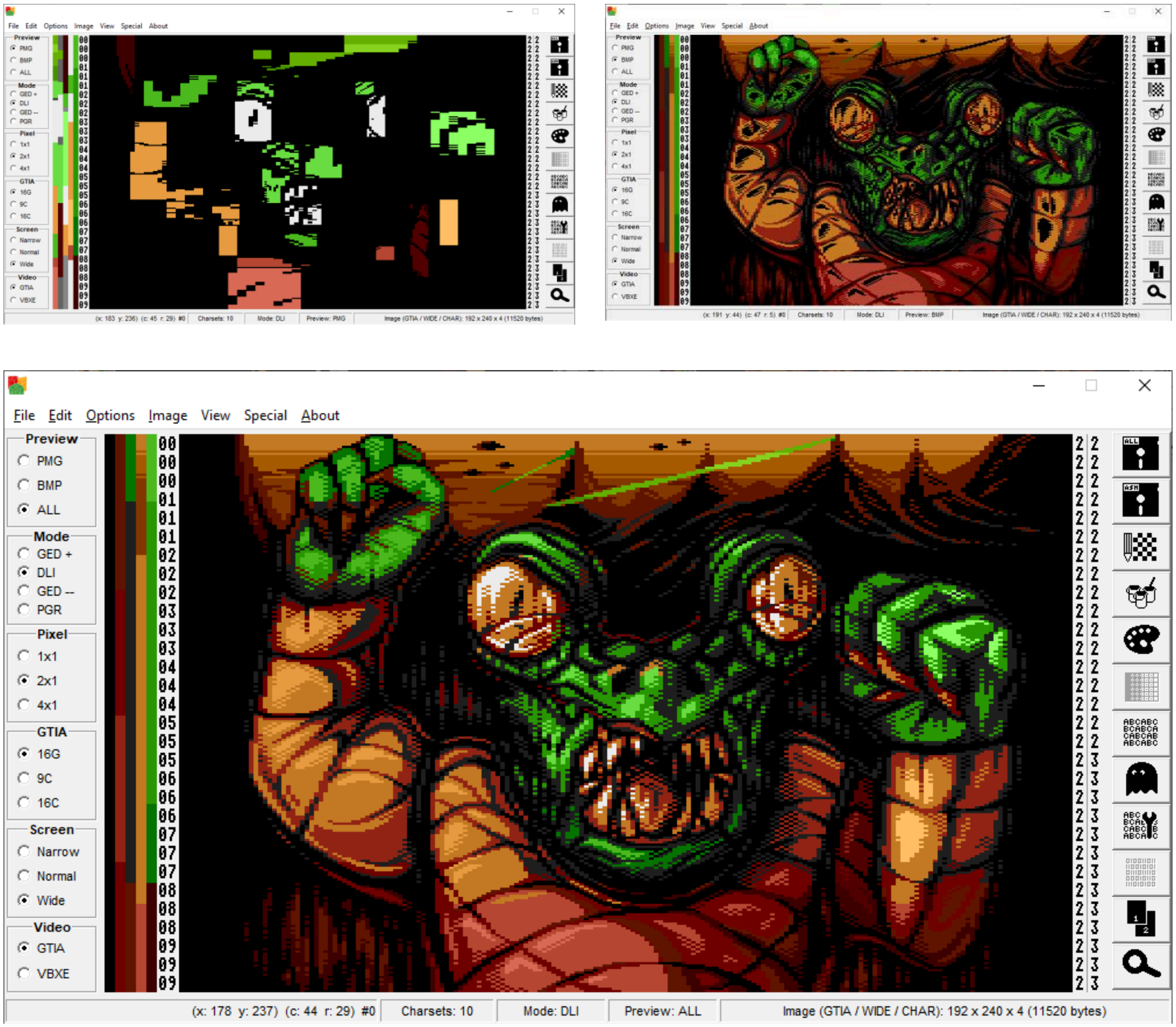
Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow [✓ Normal] Wide
Video: [✓ GTIA] VBXE

```
(y: 238)  (r: 29) | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)
```

Selecting the several graphic layers.

To demonstrate the differences between the two graphics layers of *Bitmap* graphics and *Player/Missile* graphics, I've chosen one of my favourite pictures of the great Polish pixel artist Piesiu that ranked second in the Atari XL/XE Graphics competition at Silly Venture 2014. On the next page you'll see three pictures: The picture on the top left only shows the *Player/Missile* graphics. On the picture on the top right, you can only see the *Bitmap* graphics. The big picture below displays both graphics layers.

By the way, did you notice that Piesiu did not only use four, but five colors per line? Piesiu made use of the "fifth color", which is special as it can only be displayed under certain circumstances (depending on the color number used for neighboring pixels). I will go into further detail a bit later in this book.



```
(x: 183  y: 236)  (c: 45  r: 29)  #0 | Charsets: 10 | Mode: DLI | Preview: PMG | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)
(x: 191  y: 44)  (c: 47  r: 5)  #0 | Charsets: 10 | Mode: DLI | Preview: BMP | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)
(x: 178  y: 237)  (c: 44  r: 29)  #0 | Charsets: 10 | Mode: DLI | Preview: ALL | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)
```

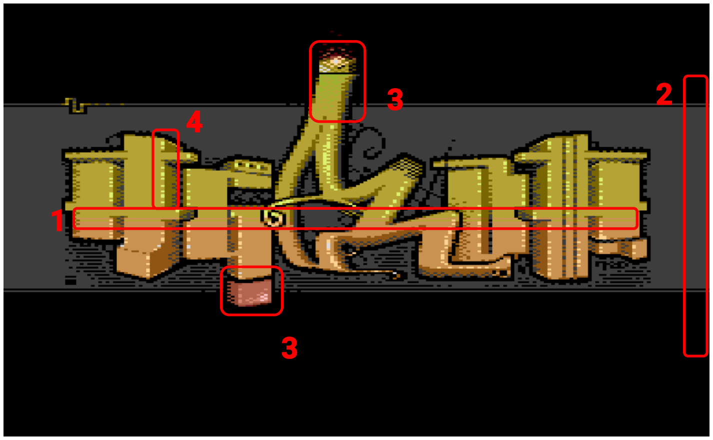
Jaggi by Piesiu using 23 colors (2014).
Man, I was so scared of that alien when I was 10 that I didn't touch *Rescue on Fractalus* for weeks!

The essential part of planning your picture

Of course, as an artist you are already visualising your picture before you start placing pixels on the screen. You have an image in your head about the general setup of the picture, of the colors for certain parts, and so on. So why mentioning this matter at all? Because when you are working with Graph2Font, you need to take an extra step. As you already know, the Atari XL/XE computer can only display four colors per line. This means that you must plan all color changes thoroughly. Moreover, you need to have a clear idea of where you want to position the *Player/Missile* graphics, as these have a limited scope of use. Planning a picture for the 8-bit Atari will have to be more thought-through than for a similar picture the Atari ST, Commodore Amiga or PC.

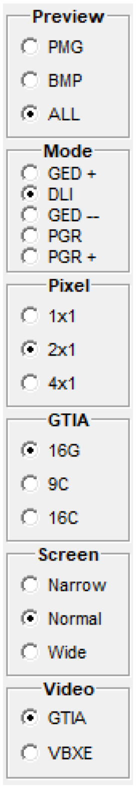
When I planned my logo for Agenda, the following points where of high importance to me:

- Each letter should feature a vertical mid-point color change (see *1* in the picture below).
- The background color in the main part of the logo should be different than the background color in the other parts of the picture (see *2*).
- Parts of the logo should appear in completely different color or with special effects (see *3*).
- There should be highlights on the edges of the letters (see *4*).



Transferring your artwork to Graph2Font

Once you've completed your planning, you can start implementing your ideas in your preferred graphics program. In my case, I used Grafx2 to draw the logo exactly as I wanted it to appear on the XL—including all the colors, details, etc., but always keeping in mind the limitations. Once the image was finished, I saved it in PNG format—although G2F also supports importing GIF and BMP files.



Preview: PMG, BMP, [✓ ALL]

Mode: GED +, [✓ DLI], GED --, PGR, PGR +

Pixel: 1x1, [✓ 2x1], 4x1

GTIA: [✓ 16G], 9C, 16C

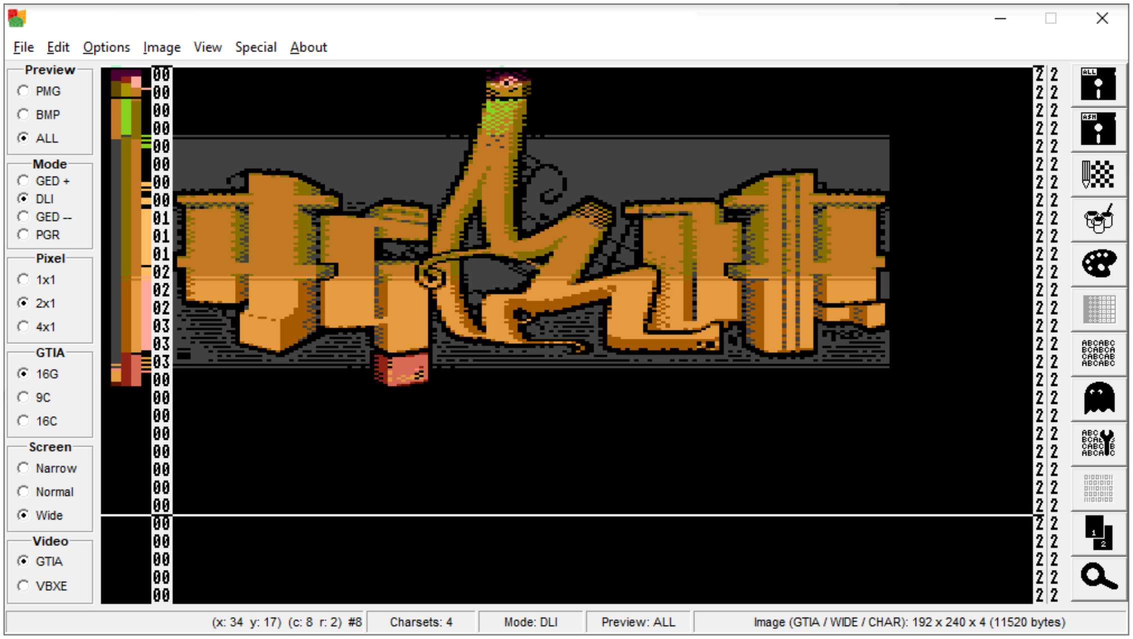
Screen: Narrow, [✓ Normal], Wide

Video: [✓ GTIA], VBXE

When you open Graph2Font you will notice there are many options with radio buttons on the left side of the screen. This can look a bit daunting so here's a quick overview of what they do.

- ***Preview*** allows to choose which part of the picture to show. Leave it on ***ALL*** for now.
- ***Mode*** lets you choose which custom character or graphic mode to use. Leave it on ***DLI***, which is the most commonly-used mode.
- ***Pixel*** allows to choose the pixel aspect ratio, which also impacts the number of colors you can use. *1x1* (High Res, up to 384×240, 2 colors), *2x1* (Low Res, up to 192×240, 4+1 colors), *4x1* (GTIA mode, max 96×240, 16 colors). We'll use is ***2x1***, the most versatile option.
- ***GTIA*** is another technical option. We'll use ***16G***.
- ***Screen*** lets you pick the screen width: *Narrow* (256 pixels in High Res), *Normal* (320 pixels in High Res), *Wide* (384 pixels in High Res). We'll use ***Normal***.
- ***Video*** specifies whether you are targetting a machine using the VBXE extension card. We are not, so we'll pick the original Atari 8-bit chipset, ***GTIA***.

You can load your PNG/GIF/BMP image into G2F simply by using the *Open* function in the *File* menu or by using the keyboard shortcut **CTRL+O**. However, I don't recommend importing your entire image into G2F, as this won't generate a satisfactory result, as you can see from the following image:

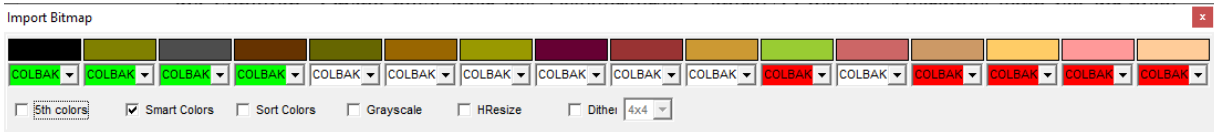


Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow Normal [✓ Wide]
Video: [✓ GTIA] VBXE

(x: 34 y: 17) (c: 8 r: 2) #8 | Charsets: 4 | Mode: DLI | Preview: ALL | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)

Importing a whole image at once may result in undesirable outcomes.

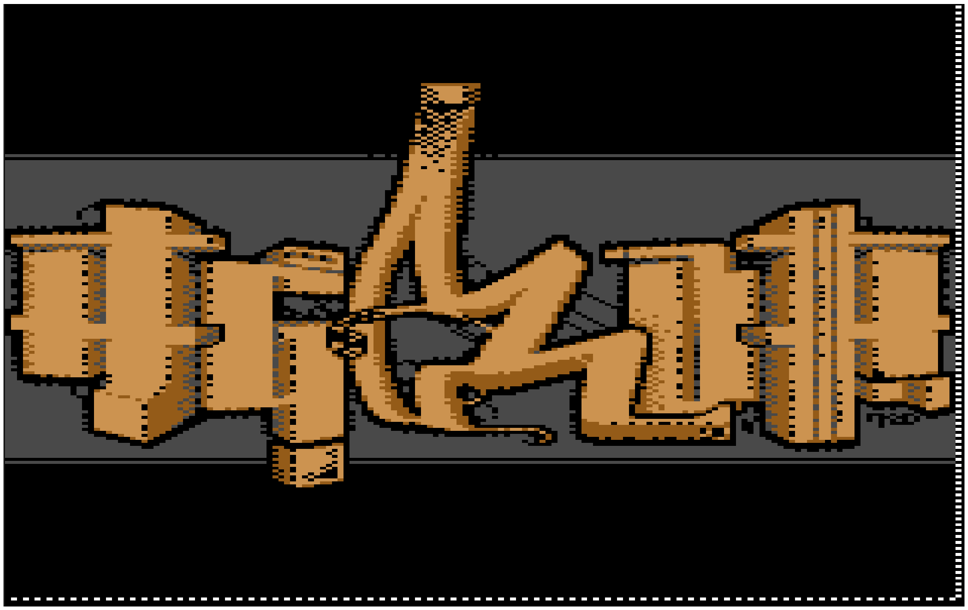
When you import a PNG image in Graph2Font, the following *Import Bitmap* panel will open. This displays the colors of the imported image. If you select *Smart Colors*, G2F will preselect them for you. The results are usually quite acceptable. You can also specify which color value should be assigned to which color register (*COLBAK* = background color, *COL0* = color 0, etc.).


Import Bitmap

COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK COLBAK

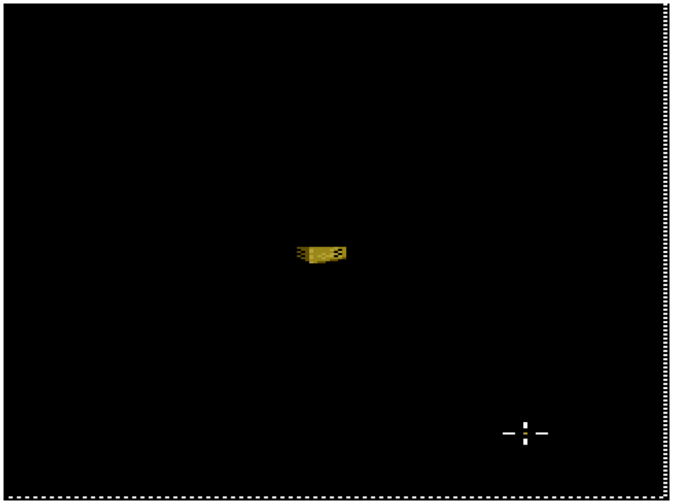
☐ 5th colors ☑ Smart Colors ☐ Sort Colors ☐ Grayscale ☐ HResize ☐ Dither 4x4

When converting the whole image at once doesn't produce an acceptable result, we need another approach, which is transferring the image in individual parts. To do this, I first masked out in Grafx2 all the parts that will be displayed as *Player/Missile* graphics, because we want to first transfer only the *Bitmap* graphics part of the image. I also reduced the logo to its four primary colors. The pure *Bitmap* graphics look like this:

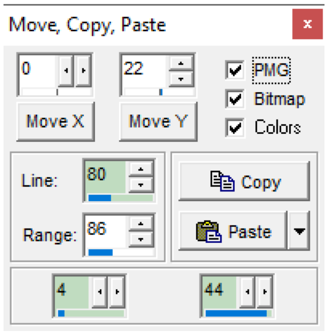


Ready for slicing: The Agenda Logo in four colors before cut into pieces for import to G2F.

I then divided the logo into sections, each of which would retain the same palette, meaning there would be no color change. I saved each of these sections individually to transfer them separately to G2F later. Some of these sections were only one or two lines high. In total, the logo was divided into 10 individual parts in this way. Here you can see the last few lines of the cube, which can be seen below the letter "G."



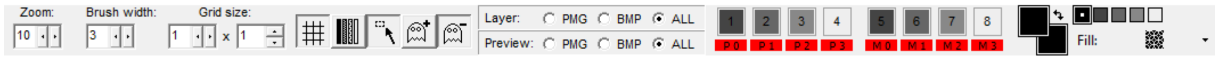
By now I had imported all the sections of my logo in G2F, but they were all in separate images. How do I get them back together? For that, I used the *Move/Copy/Paste* feature in the *Image* menu. With it, you can select a specific range of pixels, choose which layers you want (*Bitmap*, *Players/Missiles*, or all together), copy the data and paste it in a different image. With that feature and a bit of patience, it was easy to reconstruct my logo inside Graph2Font.



Pushing pixels in Graph2Font

What if you want to paint graphics from scratch in Graph2font, or make small adjustments to an imported picture without having to import it again? You can absolutely push pixels directly in Graph2Font.

As in all other art packages, the zoom function is your friend. You can enter Zoom mode with the keyboard shortcut **ALT+Z** or by clicking on the magnifying glass in the taskbar on the right. After that, a zoom window will open. What's interesting here is the toolbar at the top of it:


Zoom: 10 Brush width: 3 Grid size: 1 x 1
Layer: PMG BMP [✓ ALL]
Preview: PMG BMP [✓ ALL]

At this point, we will only focus on the functions that are important for creating *Bitmap* graphics. The other functions that refer to *Player/Missile* graphics will be described in detail later. From the left:

- Zoom: Sets the zoom factor.

- Brush width: Sets the width of your brush. Default value is 1.

- Grid icon: Displays a grid of 4×8 pixels (depending on the graphic mode you are using, preferred by some and mandatory if you want to use the "fifth color").

- Layer/Preview: *Layer* determines if pixels are set or erased only on the *P/M* layer, only on the *Bitmap* layer, or on both layers at the same time. *Preview* gives you the choice to display the *P/M* layer or *Bitmap* graphics layer only or to display both graphic layers at the same time.

- Color selection: Picks your current working color(s). By clicking the left mouse button on a color box, you will assign that color to the left mouse button. A click with the right mouse button assigns that color to that button. The black outline of the color box indicates the color currently assigned to the left mouse button and the dot in the color box indicates the same thing for the right mouse button. Hint: The keys **Q** and **W** are your friends—**Q** cycles through the palette from right to left, **W** cycles through the palette from left to right.

Colorization of your *Bitmap* graphics

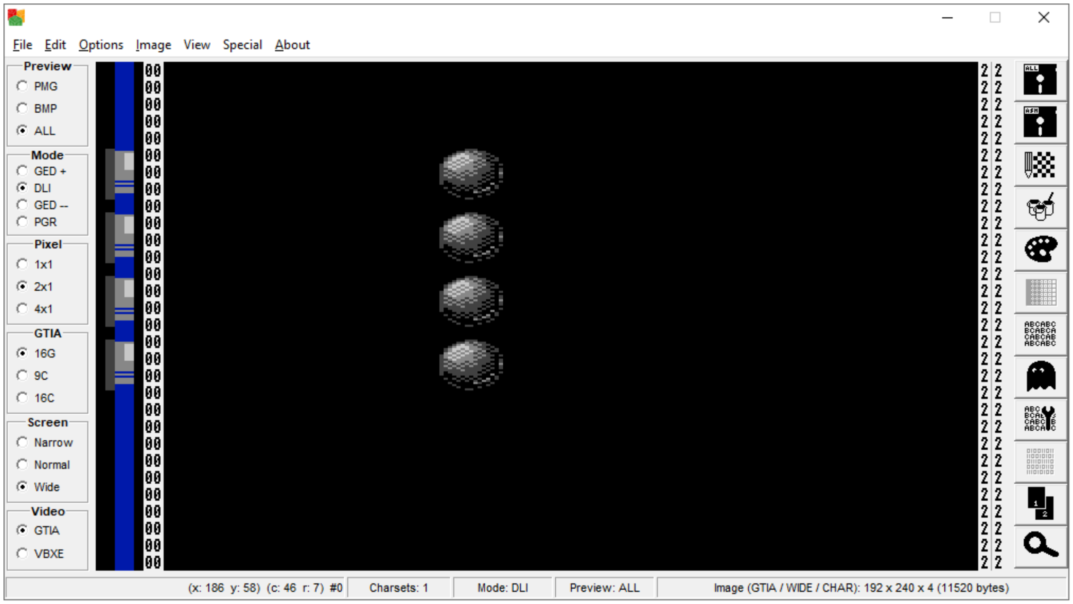
Now that you have imported your image into Graph2Font, we are going to explore how you can add more colors to it.

Addition of several palettes aka raster splits

As the Atari XL/XE computers can generally only display four colors per screen line, changing the palette on as many lines as possible is highly recommended if you want to achieve colorful images. Fortunately, these color changes can be created with ease using Graph2Font. I am going to explain how in the next few pages.

Easy Example

Let’s take a simple picture of a sphere that we want to display four times, each time in different colors:



Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow Normal [✓ Wide]
Video: [✓ GTIA] VBXE

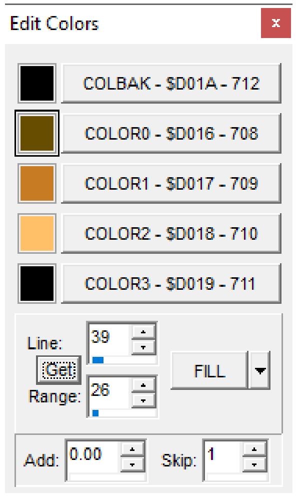
(x: 186 y: 58) (c: 46 r: 7) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)

Ready for colorization: Let’s give each sphere a different palette!

So how to colorize this picture? That's fairly easy: First you have to choose the option *Edit Colors* from the *Edit* menu (shortcut: **ALT+C**). Next, the *Edit Colors* panel opens where you can set your desired palette. The *Edit Colors* panel is arranged as follows:

Alt+C

Edit the colors of certain lines of the image



Edit Colors

COLBAK - $D01A - 712
COLOR0 - $D016 - 708
COLOR1 - $D017 - 709
COLOR2 - $D018 - 710
COLOR3 - $D019 - 711

Line: 39
Get
Range: 26
FILL

Add: 0.00 Skip: 1

In this part you see the color palette which is set in the highest screen line of the area you've selected. Left clicking on a color box or the color name will open another window with the whole color palette of the Atari XL/XE, from where you can pick a color. Please keep in mind that if you want to change a color, don't close the window before you have chosen *FILL* or else your color choice won't have any effect.

The number next to *Line* shows the starting screen line of the current palette, *Range* shows the number of screen lines that will be affected by the color changes you are setting up. *FILL* lets you apply a color change. *Get* grabs the nearest *Bitmap* data and often helps find the right settings for *Line* and *Range* (just try it!).

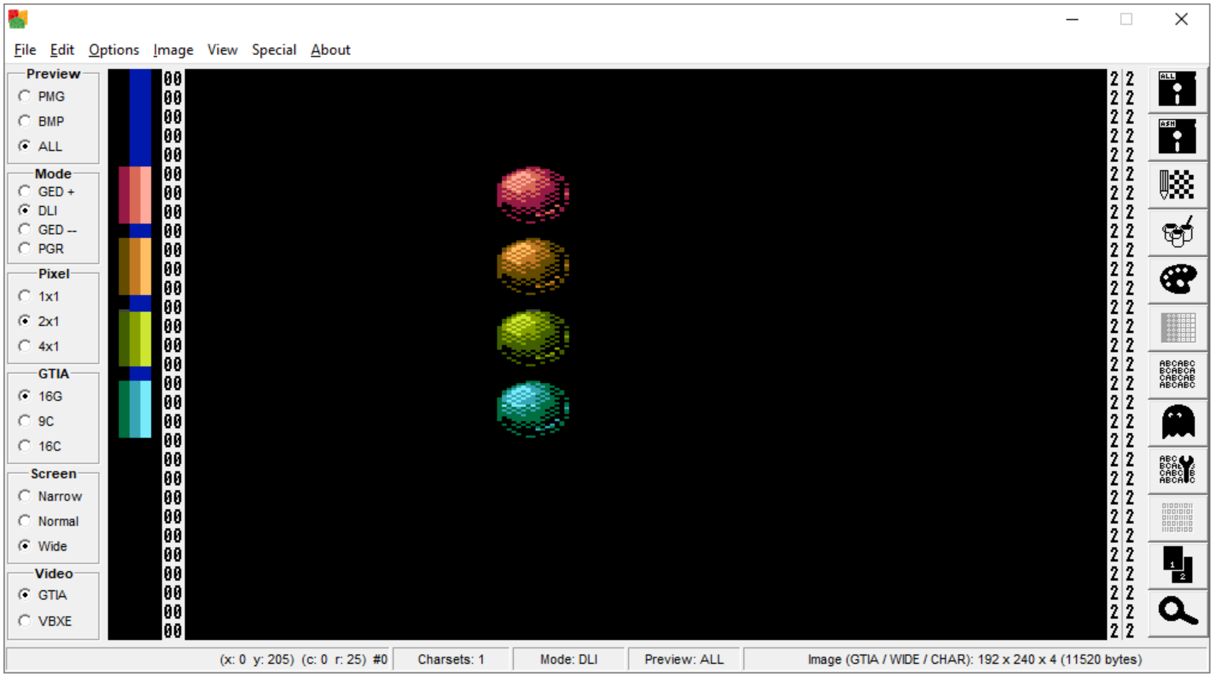
Alternatively you can directly click the image with the left mouse button to select the starting screen line (the number displayed next to the *Line* option will be affected immediately by this). The screen area that will be affected by the color changes is contained between the two horizontal white lines.

Finally, there are the two functions *Add* and *Skip* at the bottom of the window. These enable you to design nice color ramps starting at the color you've chosen. Do you remember the raster bars on page17? These were done with the following settings: select *COLBAK*, *Line* = 0, *Range* = 239, *Add* = 1.00 and *Skip* = 1.

Back to the four grey spheres, in order to colorize them we need to follow the following steps:

1. Opening the *Edit Colors* panel (**ALT+C**).
2. Marking the vertical starting position of the palette (highest screen line) with the mouse (or by entering a number at *Line*).
3. Entering the number of lines affected by the color change at *Range*;

4. Picking the color slot that should be changed and picking a color from the palette, each time followed by clicking on *FILL*.



Preview: PMG BMP [✓ ALL]

Mode: GED + [✓ DLI] GED -- PGR

Pixel: 1x1 [✓ 2x1] 4x1

GTIA: [✓ 16G] 9C 16C

Screen: Narrow Normal [✓ Wide]

Video: [✓ GTIA] VBXE

(x: 0 y: 205) (c: 0 r: 25) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)

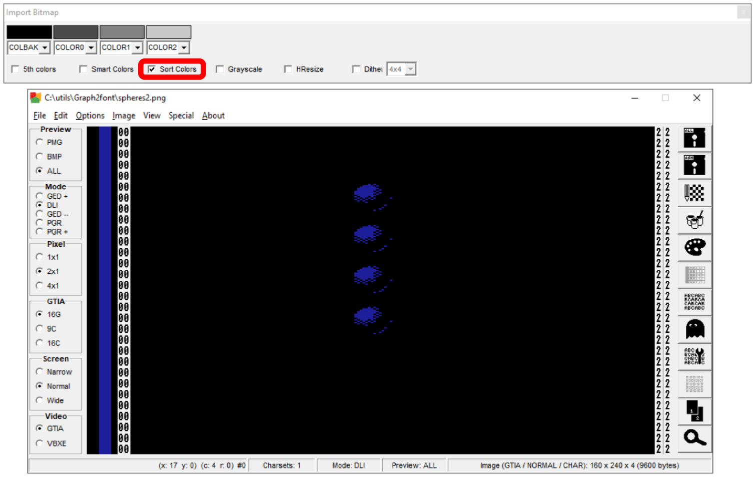
Colored spheres with four different color palettes shown on the left side of the screen.

Step-by-step process

Understanding the *Edit Colors* process is key to mastering Graph2Font, so let's look in further detail at the steps required to recolor the first sphere:

1. This is our starting point: the sphere picture as it was imported from Grafx2, using the default parameters except for having the *Sort Colors* option ticked on so the colors are nicely sorted from dark to light. Don't worry if the colors look wrong here, the underlying color data is correct. In this case, I found

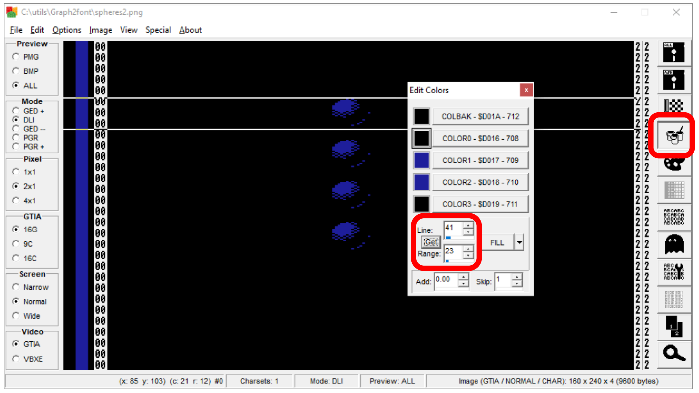
that using *Smart Colors* looked better at first, but color indexes were not used in a consistent way which made it harder to recolor the picture later.



[✓ Sort Colors]

(x: 17 y: 0) (c: 4 r: 0) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

2. I press **Alt+C** or click the icon with the three little paint buckets; the *Edit Colors* panel shows up.
3. I enter the following parameters to fully cover the lines used by the top sphere: *Line*: 41, *Range*: 23. I can see the whole sphere is contained between the two white lines.

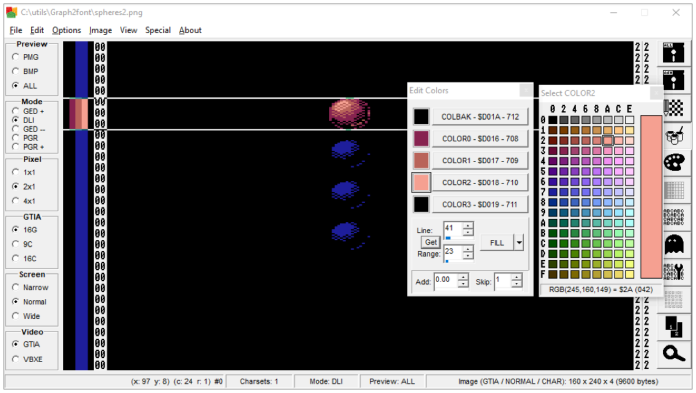


COLBAK - $D01A - 712
COLOR0 - $D016 - 708
COLOR1 - $D017 - 709
COLOR2 - $D018 - 710
COLOR3 - $D019 - 711
Line: 41
Range: 23
Add: 0.00
Skip: 1

(x: 85 y: 103) (c: 21 r: 12) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

4. I can now update the four colors of the sphere by clicking on each color box:

- I keep *COLBAK* (the background color) as it is, the pure black color $00
- For *COLOR0* I pick color $32 (second column, fourth row down) and press *FILL*.
- For *COLOR1* I pick color $26 (fourth column, third row down) and press *FILL*.
- For *COLOR2* I pick color $2A (sixth column, third row down) and press *FILL*.



Preview: [✓ ALL]; Mode: [✓ DLI]; Pixel: [✓ 2x1]; GTIA: [✓ 16G]; Screen: [✓ Normal]; Video: [✓ GTIA]

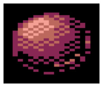
COLBAK - $D01A - 712
COLOR0 - $D016 - 708
COLOR1 - $D017 - 709
COLOR2 - $D018 - 710
COLOR3 - $D019 - 711

Line: 41
Range: 23
Add: 0.00
Skip: 1

RGB(245,160,149) = $2A (042)

(x: 97 y: 8) (c: 24 r: 1) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

And that's it, really! I can now close the *Edit Colors* panel and admire my newly-recolored sphere.



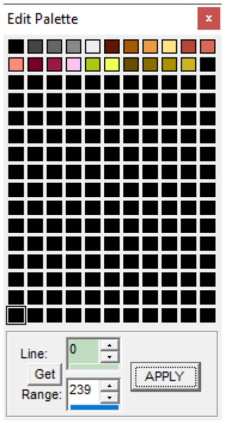
Now it's time to learn about the colored areas on the left side of the screen (on the right side of the menu bar with *Preview*, *Mode* and *Pixel*). This gives you an overview of the color palette set for the screen line opposite. On the far left is the background color (*COLBAK*), on the far right is the 'fifth color' (*COL3*) which will be explained later (obviously, it is not set in the picture above).

Edit palette

Once you finished your picture with several color palettes, you may want to adjust one or more colors. Take for instance the picture with the colored spheres shown above. If you just want to change the black background color to let's say a dark blue tone, you don't need to adjust every single of the nine color palettes (4 palettes for the balls, 5 palettes for the spaces above, below and in between). You can simply use the *Edit Palette* function in the *Edit* menu (shortcut: **Alt+L**).

Alt+L

Calls the *Edit Palette* function



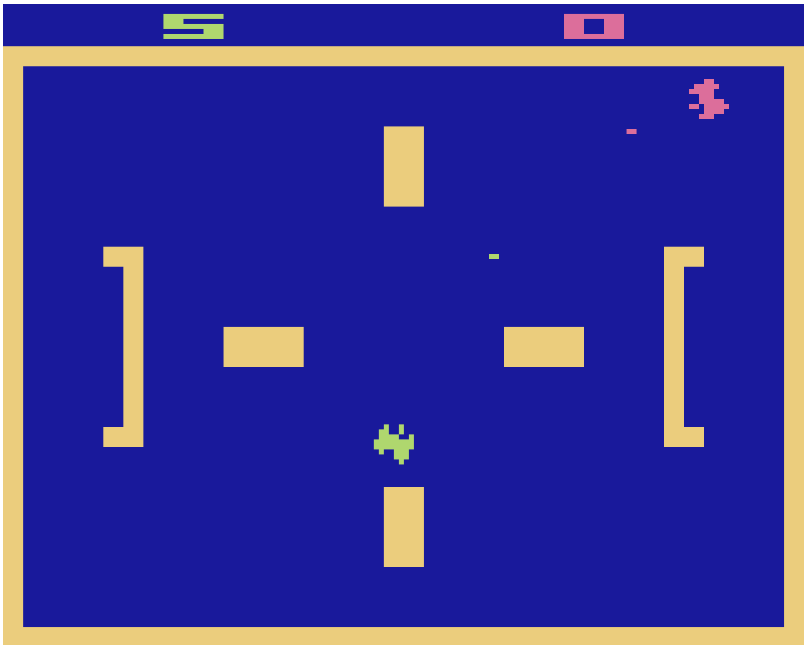
The *Edit Palette* panel displays all the colors used in the picture, whether they are colors used in the *Bitmap* part of your image or by *Player/Missile* graphics. Click on the color you want to change and pick a color from the color palette window that pops up.

The value entered at *Line* marks the first line of the screen that will be affected by a change. *Range* defines the number of lines after the first line that will be included.

You surely will find out what the *APPLY* button is about.

Players* and *Missiles

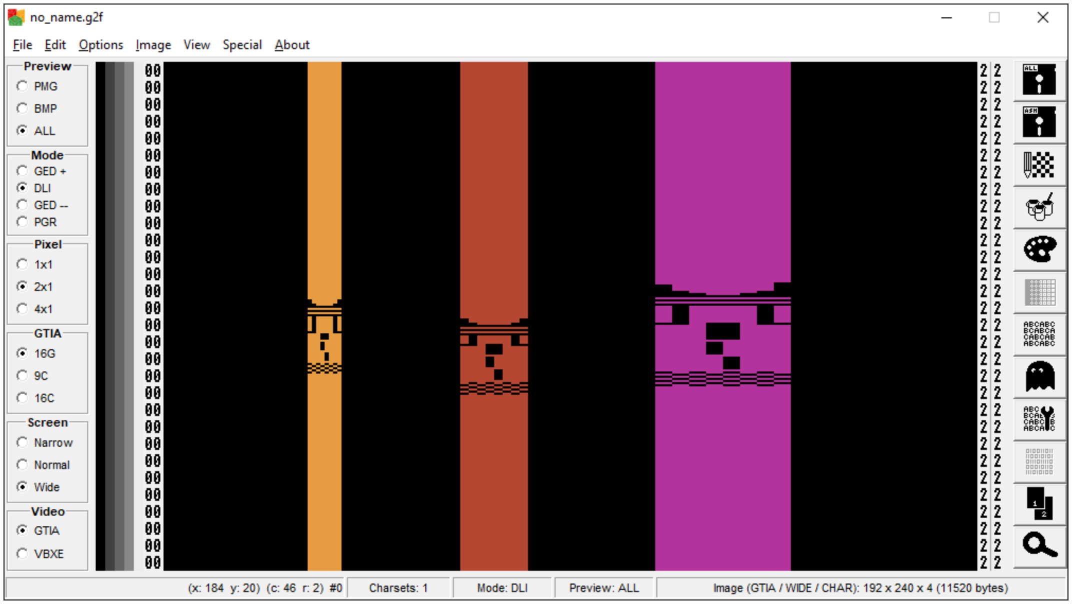
The concept of *Player* and *Missile* objects goes all the way back to the humble Atari 2600; they were basically sprites before the term was invented. If you look at an early Atari 2600 game like *Combat* (1977), then the way the screen is drawn and what the different elements are called makes perfect sense.


Combat on the Atari 2600

There are two tanks on screen (the *Players*). Each can shoot projectiles (the *Missiles*). The rest of the screen is static (the *Playfield*). While the *Player* and *Missile* objects are generally displayed above the *Playfield*, they can also be displayed behind it.

Characteristics of *Players*

Back on the Atari 8-bit, the picture below highlights the main characteristics of *Player* objects:



Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow Normal [✓ Wide]
Video: [✓ GTIA] VBXE

(x: 184 y: 20) (c: 46 r: 2) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)

- **Amount:** There's generally a maximum on 4 *Players* objects displayed at once.
- **Height**: Each *Player* can have a vertical size of up to 240 pixels (i.e. the full height of the screen).
- **Width**: A *Player* sprite is always 8-pixel wide, but the scale of these pixels can be increased if needed. On the image above, the yellow *Player* sprite on the left has pixels with a horizontal scale of 1 (covering 8 pixels on the image), the red *Player* sprite in the middle has pixels with a horizontal scale of 2 (covering 16 pixels on the image), and the purple *Player* sprite on the right has pixels with a horizontal scale of 4 (covering 32 pixels on the image).

    - Each pixel of a *Player* sprite in **regular scale** appears as **1-pixel wide** on the image.
    - In **2x scale**, it appears as **2-pixel wide** on the image.
    - In **4x scale**, it appears as **4-pixel wide** on the image.
- **Color**: each *Player* sprite is limited to a single color per line, but that color is completely independent from the colors used in the rest of the image.

But there's more to know about *Players* that is not obvious yet:

- **Horizontal position**: The 1x wide *Player* can start at any X coordinate you assign to it. However, the 2x-wide and 4x-wide *Players* have a lower precision. The 2x-wide *Player* can only start at X coordinates divisible by 2 (2, 4, 6, 8, ...). The 4x-wide *Player* can only start at X coordinates divisible by 4 (4, 8, 12, 16, 20, ...). You can change the X coordinates of a *Player* at every line, so for instance the first line could start at X = 2, and the second line at X = 150.

- **Horizontal scale**: Each *Player* can change its horizontal scale (single, double, quadruple) on every line.

- **Color**: Each *Player* can change its color on every line.

- **Overlay/underlay**: Each *Player* as a whole can be displayed above or below *Bitmap* graphics. The overlay/underlay setting is set for the whole screen which means you cannot change it line by line.

- **Mixing *Player* colors**: It's possible to mix the colors of two *Players*. A real pro feature (albeit of limited use) that I will investigate later.

Characteristics of ***Missiles***

Missiles behave very much like *Players* except they are narrower.

- **Amount:** There's a maximum on 4 *Missile* objects displayed at once.
- **Height**: Each *Missile* can have a vertical size of up to 240 pixels (i.e. the full height of the screen).
- **Width**: A *Missile* sprite is always 2-pixel wide (1/4th of a *Player*), but the scale of these pixels can be increased if needed, like for *Players*. 2x scale covers 4 pixels, 4x scale covers 8 pixels.
- **Color**: each *Missile* sprite is limited to a single color per line. **If a *Player* and a *Missile* from the same set are displayed on the same line, they need to use the same color**.

Bringing ***Players*** to the screen

"Painting" with *Players* will be a new experience for you since you cannot simply choose the zoom function and start pushing pixels on screen. Instead, it is necessary to first place the *Player* or *Missile* sprites at specific screen coordinates and to define their horizontal and vertical sizes. Initially, the sprites are plain blocks of color, and only from that point you can set or delete single pixels. Painting with *Players/Missiles* in Graph2Font always requires the following three steps:

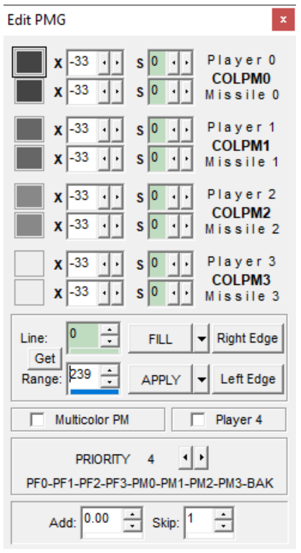
1. Exact positioning of the *Player/Missile* object on the screen.
2. Defining the horizontal and vertical size of the *Player/Missile* object (width and height).
3. Refining the shape of the *Player/Missile object* with the zoom function.

1. Positioning of the *Player/Missile* on the screen

To position one of the *Players* on the screen, you must select the function *Edit PMG* in the *Edit* menu. Alternatively, you can use the keyboard shortcut **ALT+P** or click on the ghost icon on the toolbar on the right side of the Graph2Font window. The *Player/Missile Edit* panel will appear. We've already seen it earlier when we've learnt about the general principle of *Player/Missile* graphics.

Alt+P

Opens the *Player/Missile* panel


Edit PMG

x -33 s 0 Player 0
COLPM0
x -33 s 0 Missile 0

x -33 s 0 Player 1
COLPM1
x -33 s 0 Missile 1

x -33 s 0 Player 2
COLPM2
x -33 s 0 Missile 2

x -33 s 0 Player 3
COLPM3
x -33 s 0 Missile 3

Line: 0 FILL Right Edge
Get
Range: 239 APPLY Left Edge

Multicolor PM Player 4

PRIORITY 4
PF0-PF1-PF2-PF3-PM0-PM1-PM2-PM3-BAK

Add: 0.00 Skip: 1

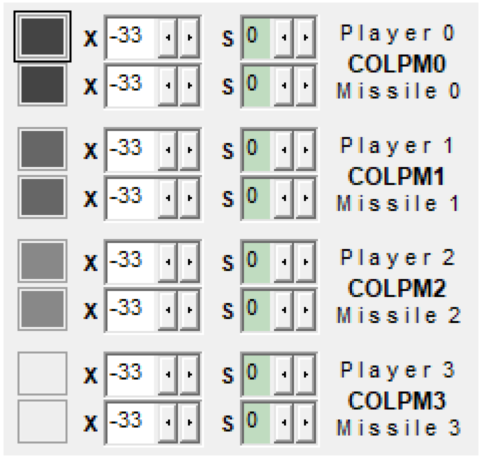
Set color, horizontal position and horizontal size of *Player/Missile*.

Set vertical position, color filling and add/delete a *Player/Missile*.

Various options.

Let's have a closer look at the details of the *Player/Missile* Edit menu:

You can select the color of a *Player* or *Missile* by clicking on the color box on the left. The color of a *Player* and the color of its *Missile* is always identical.

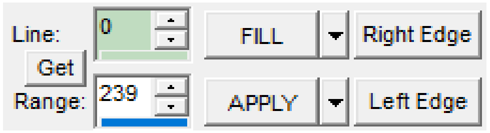


X -33 S 0 Player 0
COLPM0
X -33 S 0 Missile 0

X -33 S 0 Player 1
COLPM1
X -33 S 0 Missile 1

X -33 S 0 Player 2
COLPM2
X -33 S 0 Missile 2

X -33 S 0 Player 3
COLPM3
X -33 S 0 Missile 3



Line: 0
Range: 239

X defines the horizontal position of *Players* and *Missiles*. The horizontal position is -33 by default, which is outside the visible screen on the left. You can also position them by clicking with the mouse on the screen.

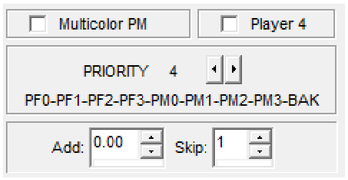
S sets the horizonal scale of the *Player* (0 = 1x, 2 = 2x, 4 = 4x).

Line defines the first screen line in which your *Player/Missile* will be displayed. Alternatively, you can position them by left-clicking with the mouse on the image.

By entering a value next to ***Range*** you can set the vertical size of the *Player/Missile*. In the picture above the *Player* starts at line 0 and is 239 lines high, which means it is shown from the upper to the lower border of the screen.

FILL paints your *Player/Missile* object with the color you've chosen before. Please note that you need to click on this button for the color change to be effective!

APPLY accepts all your previous settings and finally puts the *Player/Missile* on screen. This should be your final working step! The drop down menu on the ***APPLY*** button lets you specify how you want modify the image: *Change* is the default setting which means that if no *Player/Missile* is set in your selected area, ***APPLY*** will set it; if a *Player/Missile* is already set, ***APPLY*** will delete it. You can experiment with ***CLEAR***, ***FILL*** and ***DELETE*** – just take your time to assess the different options!



☐ Multicolor PM ☐ Player 4

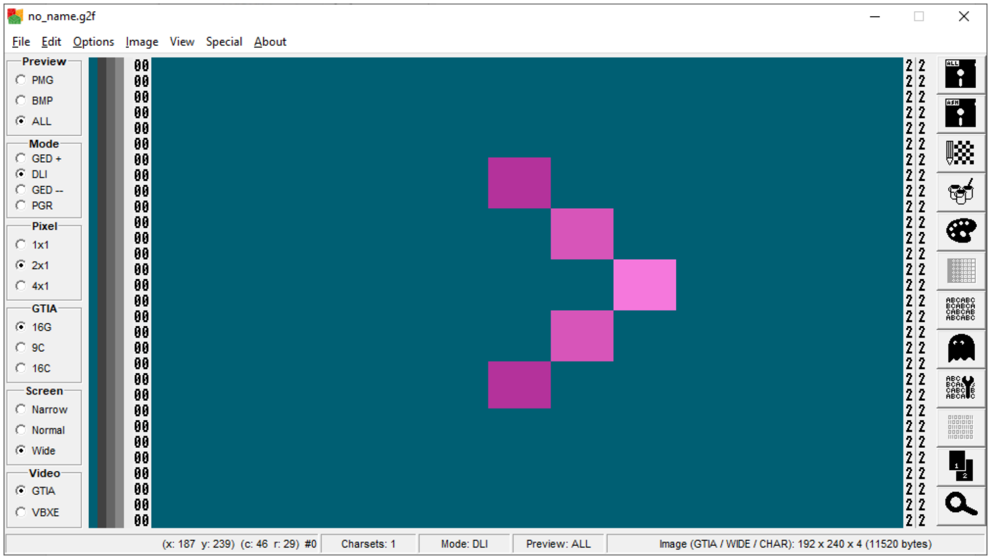
PRIORITY 4

PF0-PF1-PF2-PF3-PM0-PM1-PM2-PM3-BAK

Add: 0.00 Skip: 1

The functions *Add* and *Skip* have already been covered (remember the colorful raster bars earlier?) and enable you to fill *Players/Missiles* with gradients. The *Multicolor PM* function will be explained as a "Pro Feature" at a later point (p.48).

There's one important feature of *Player/Missile* graphics that greatly enhances their value: Each *Player* can only be displayed once per screen line, but the horizontal position, the pixel scale and the color of that *Player* can vary from line to line. A *Player* can be displayed in double size and a red color on the left border on lines 0 to 20 and in quadruple size and a blue color on the right border on lines 40 to 60. Let's demonstrate this with the following picture which shows a single *Player* object with various horizontal positions and colors:



Preview: PMG BMP [✓ ALL]

Mode: GED + [✓ DLI] GED -- PGR

Pixel: 1x1 [✓ 2x1] 4x1

GTIA: [✓ 16G] 9C 16C

Screen: Narrow Normal [✓ Wide]

Video: [✓ GTIA] VBXE

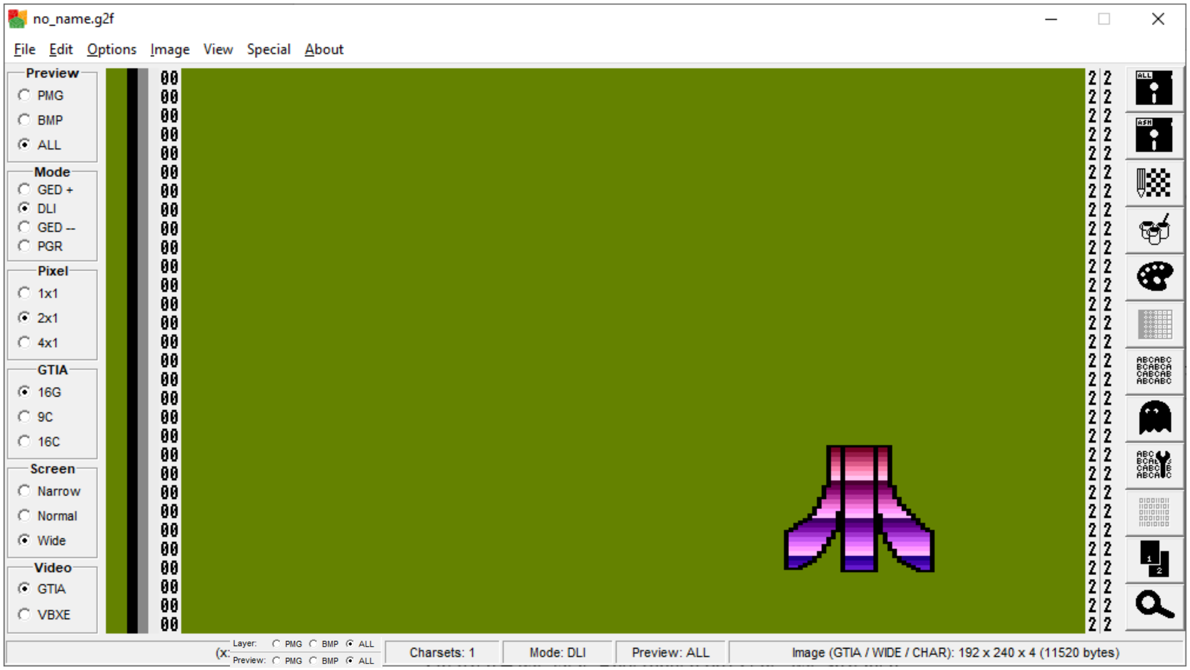
(x: 187 y: 239) (c: 46 r: 29) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)

Player/Missile can change width, position and color for line to line.

To achieve this, you just need to repeat the steps described above—determining the X and Y coordinates and pressing *FILL* and *APPLY* each time at the desired position.

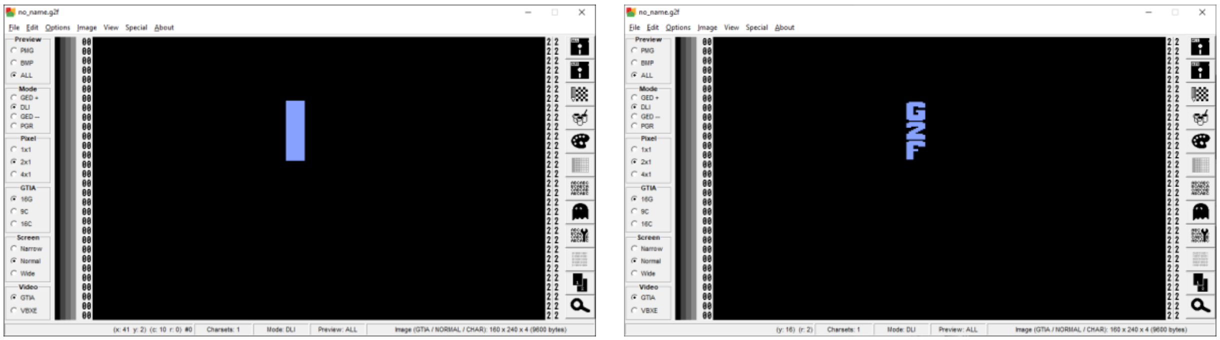
2. Shaping your *Player/Missile* objects

The default sprites created in the *Player/Missile* menu are always rectangular and unattractive. Often you'll want to give them a custom shape. However, there are cases where you might want to keep them in their original shape, such as when displaying a *Player* (or *Missile*) in the background of a *Bitmap* graphic and having the *Player* colors show through "holes" in that foreground. For example, on the image below the fill of the Atari logo is a 64-pixel-wide square *Player* showing through a *Bitmap* graphic box drawn in *COL0* (it has the same color as the background color, $D6 = 218), while the logo's black outline is drawn as a *Bitmap* graphic in *COL1*.



Preview: PMG, BMP, [✓ ALL]
Mode: GED +, [✓ DLI], GED --, PGR
Pixel: 1x1, [✓ 2x1], 4x1
GTIA: [✓ 16G], 9C, 16C
Screen: Narrow, Normal, [✓ Wide]
Video: [✓ GTIA], VBXE

Layer: PMG, BMP, [✓ ALL]
Preview: PMG, BMP, [✓ ALL]
Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / WIDE / CHAR): 192 x 240 x 4 (11520 bytes)

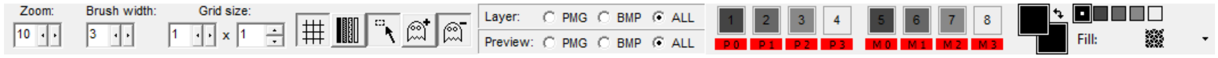


(x: 41 y: 2) (c: 10 r: 0) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

(y: 16) (r: 2) | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

From block to logo: Shaping your *Player/Missile* sprites

In most cases, you'll want to give your *Player/Missile* sprites a custom shape. Because the pixels of the *Player/Missile* sprites can only be set and deleted in zoom mode, you must switch to the Zoom window first by clicking on the icon with the magnifying glass in the toolbar or by pressing **Alt+Z**. Just as a quick reminder, here is the toolbar of the zoom window once again:



Zoom: 10 | Brush width: 3 | Grid size: 1 x 1 | Layer: ( ) PMG ( ) BMP [✓ ALL] | Preview: ( ) PMG ( ) BMP [✓ ALL] | 1 2 3 4 5 6 7 8 | P 0 P 1 P 2 P 3 M 0 M 1 M 2 M 3 | Fill:

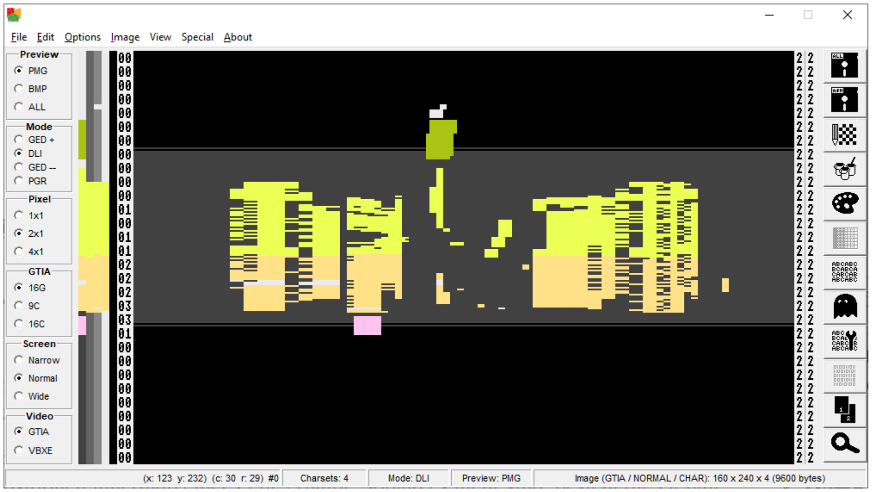
For shaping *Player/Missile* blocks, the following functions of the toolbar are of importance:

- Draw/Delete *Player/Missile* pixels: Unlike with *Bitmap* pixels, you must first select whether you want to add or delete pixels on your *Player* or *Missile* objects. The ghost icon with the plus sign is meant for drawing pixels (shortcut: **O**), while the ghost icon with the minus sign is meant for deleting pixels (shortcut: **A**). You can delete or set multiple pixels at once by holding down the mouse button. Unlike when drawing *Bitmap* pixels and unlike the way nearly all other graphics programs behave, here the Zoom window and the actual image data only update when you release the mouse button, which can be quite confusing.

- Layer/Preview: Layer: ( ) PMG ( ) BMP [✓ ALL] Preview: ( ) PMG ( ) BMP [✓ ALL] Select *PMG* as a *Layer* option to ensure that your pixels are only changed at the *Player/Missile* graphic level and not at the *Bitmap* graphic level.

- Show/Hide *Players/Missiles*: This control allows to show or hide specific *Players* or *Missiles*. You can achieve the same effect by pressing the keys **1** to **8**.

Customizing the *Player/Missile* sprites might not sound too exciting but it will take up a lot of your time. On the Agenda logo, the *Players/Missiles* looked like this once fully customized:



Preview: [✓ PMG] BMP ALL
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow [✓ Normal] Wide
Video: [✓ GTIA] VBXE

(x: 123 y: 232) (c: 30 r: 29) #0 | Charsets: 4 | Mode: DLI | Preview: PMG | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

Although it may not look like it, the *Players* and *Missiles* here were used solely to create the highlights on the edges of the letters, to get the green dithering at the top of the letter E, to create the brightest color in the cuboid below the letter G, and to create a few white pixels in the fire effect above the E.

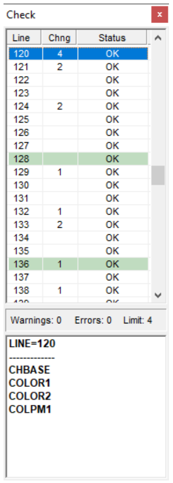
The limits of changes per line

I've explained earlier that it's possible to change to color palette for the *Bitmap* graphics and the position and color of the *Player/Missile* graphics from screen line to screen line. But that's only half the truth, because these changes need to happen in the non-visible area of the screen (in the right or left border during the Horizontal Blank Interrupt). Nothing would ruin an image more than an a color change taking too long and ending up happening in the middle of the next line!

These changes are quite taxing for the Atari 8-bit hardware so they need to be used sparingly. In practice, **no more than four instances of either color change, *Player/Missile* position change, or *Player/Missile* size change are allowed on a single screen line**. If there are more than four changes, the display will not keep up and your picture may not show correctly on the real hardware (you can test this anytime by saving your picture as a XEX file in Graph2Font and load this file in your favourite emulator). In order to check if your picture can be displayed without any problems on an Atari XL/XE, you can select *Check* from the *Options* menu (shortcut: **CTRL+E**).

Ctrl+E

Checks for warnings and errors



Line	Chng	Status
120	4	OK
121	2	OK
122		OK
123		OK
124	2	OK
125		OK
126		OK
127		OK
128		OK
129	1	OK
130		OK
131		OK
132	1	OK
133	2	OK
134		OK
135		OK
136	1	OK
137		OK
138	1	OK

Warnings: 0 Errors: 0 Limit: 4

```
LINE=120
-------------
CHBASE
COLOR1
COLOR2
COLPM1
```

The *Check* panel is mostly self-explanatory: Looking at the headers, *Line* refers to the screen line (0 to 240), *Chng* displays the number of changes in this particular screen line and *Status* gives a warning message as soon as there are too many changes.

Below the table, the total amount of warnings and errors on all 240 lines is displayed. This makes your work a bit easier, as you don't have to check the whole table line by line.

By clicking on a line from the table, you can see all the changes that happen on that particular line. Some examples are:

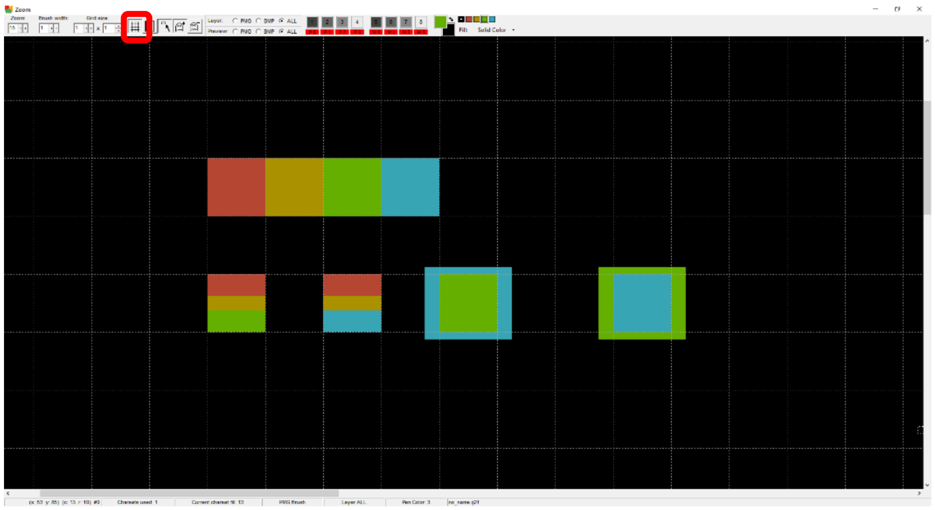
- *COLOR1* = change of (*Bitmap*) color 1
- *COLPM0* = color change of *Player/Missile* 0
- *HPOSM0* = change of the horizontal position of *Missile* 0

Pro Feature: The fifth color

Since we learnt about the fundamentals of Atari XL/XE graphics, we know that it is possible to show *Bitmap* graphics in four colors per screen line. But that's only if you want to make it easy for yourself and if you don't want to invest even more in planning and fine adjustments—and maybe even suffer more frustration ;-)

If you are willing to walk the extra mile, you can use the "fifth color" feature to make your picture even more colorful. We've already seen with Piesiu's *Jaggi* picture on page 21 that G2F experts are making use of this additional color. The two pictures by Odyn1ec from Silly Venture 2014 and 2016 on pages 5 and 6 use it as well, as you can tell from the color index on the left of the picture which has data in the fifth color row.

So, what is so special about using that fifth color? It's the fact that this color cannot be displayed in the same region as another color. To be exact: pixels in *COL3* (as the fifth color is named in the color menu) cannot be used in the same 4×8 pixels block if even a single pixel in *COL2* is present there. The blocks in which these incompatibilities appear, are static—they can be found at the same spots on each screen, like a grid. To see them, just select *Grid* in the *Zoom* menu:

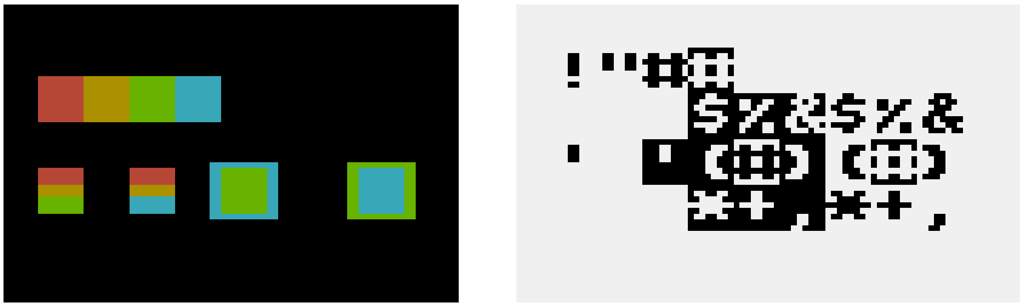


Using the fifth color requires extra planning… and a higher tolerance for frustration (Graph2Font zoom view).

You can see the limits of using the fifth color (in this case, cyan) clearly in the picture above: Everything is fine as long *COL3* is not used in the same 4x8 pixels block as the neighboring color (*COL2*, green). As soon as a single pixel in *COL3* is placed in the same block in which a pixel of *COL2* is already present, the *COL2* content will be recolored to *COL3*.

Why is that? For an answer we must take a look into the engine room of Graph2Font, even though that might not look too meaningful for us pixel artists. Underneath all the shiny colors on screen, the *Bitmap* graphics layer of Graph2Font is actually made of the characters of a custom font! Suddenly that "Graph2font" name makes a lot more sense too. If you want to lift the curtain and see your picture as a grid of characters, just select *Show Chars* in the *Options* menu. Alternatively, you can use the keyboard shortcut **CTRL+H** or click on the "ABCABC" icon on the toolbar on the right side of the screen.

Let's compare our test picture from above as a *Bitmap* picture (on the left) and a charset picture (on the right):


Welcome to the machine: Displaying *Bitmap* graphics as font characters.

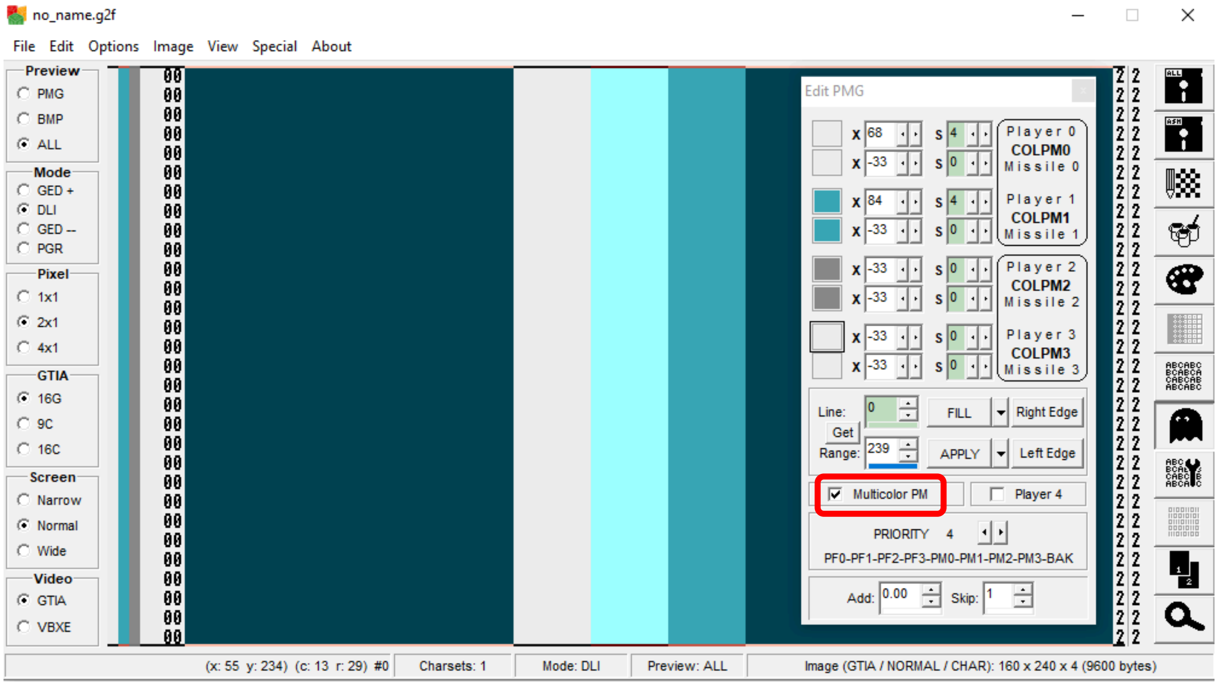
What do we see?

- The black background (*COLBAK*) is represented by non-inverted space characters.
- The red block in *COL0* is represented by a non-inverted **!** character.
- The yellow block in *COL1* is represented by a non-inverted **"** character.
- The green block in *COL2* is represented by a non-inverted **#** character.

- The blue block in *COL3* (fifth color) is represented by an inverted # character. As # cannot be inverted (*COL3*) and non-inverted (*COL2*) at the same time, it's not possible to display *COL2* and *COL3* within the same character field.
- All areas in which *COL3* is (partially) used are represented by inverted characters. Just look at the characters `$ % & ( )` and `* + ,` which are used in their inverted and in their non- inverted form. I guess you get the underlying idea now!

Pro Feature: Multicolor *Players/Missiles*

Multicolor *Players/Missiles* are another pro feature that will spice up your picture even more—although the practical use is rather limited (I have yet to find a picture using this function). To use Multicolor *Players/ Missiles*, you have to select *Multicolor PM* in the *Edit PMG* window (shortcut: **ALT+P**) and click on *APPLY*.



no_name.g2f
File Edit Options Image View Special About
Preview
Mode
Pixel
GTIA
Screen
Video
Edit PMG
Player 0 COLPM0 Missile 0
Player 1 COLPM1 Missile 1
Player 2 COLPM2 Missile 2
Player 3 COLPM3 Missile 3
FILL
Right Edge
Get
APPLY
Left Edge

Preview: PMG BMP [✓ ALL]
Mode: GED + [✓ DLI] GED -- PGR
Pixel: 1x1 [✓ 2x1] 4x1
GTIA: [✓ 16G] 9C 16C
Screen: Narrow [✓ Normal] Wide
Video: [✓ GTIA] VBXE

Player 0: x 68, s 4; Missile 0: x -33, s 0
Player 1: x 84, s 4; Missile 1: x -33, s 0
Player 2: x -33, s 0; Missile 2: x -33, s 0
Player 3: x -33, s 0; Missile 3: x -33, s 0

Line: 0
Range: 239
[✓ Multicolor PM] Player 4
PRIORITY 4
PF0-PF1-PF2-PF3-PM0-PM1-PM2-PM3-BAK
Add: 0.00 Skip: 1

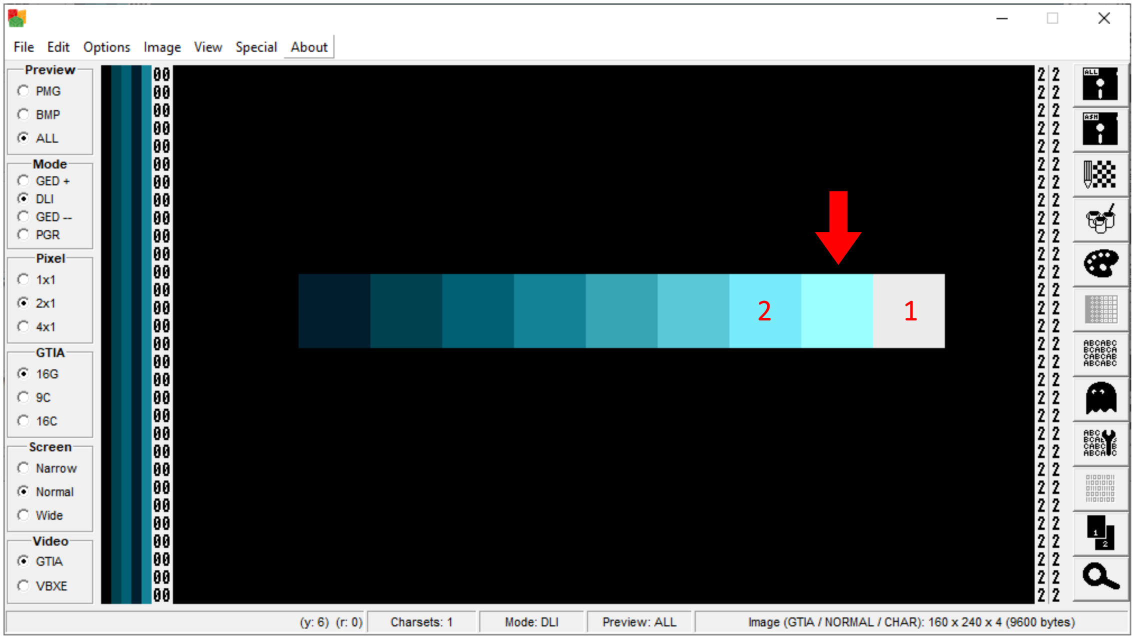
(x: 55 y: 234) (c: 13 r: 29) #0 | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

Multicolor *Player/Missile*: Two overlapping *Players* generate an additional *Player* color.

Once *APPLY* has been pressed, you will notice two boxes with rounded edges around *COLPM0 / COLPM1* and *COLPM2 / COLPM3* in the top right of the *Edit PMG* panel. This indicates that the Multicolor *Player/ Missile* effect is only available between *COLPM0 / COLPM1* and *COLPM2 / COLPM3* and not between COLPM0 and COLPM2 or any other combination.

What is the effect of Multicolor *Players/Missiles* in practice? It lets us place two *Players* from the same "boxed" group at the same position. Wherever they overlap, their colors get "logically ORed" which results in another color. Thus, a *Player* object can have two colors within the same screen line: its original color and the color that results from overlapping with another *Player*.

Just have a look at the picture below: The white box on the far right marked *1* is a quadruple-sized *Player* (64 pixels), the third box from the right marked *2* is another quadruple-sized *Player* (64 pixels). They overlap over 32 pixels in the middle, which results in the mixed color tone of the second box from the right, marked by the red arrow.



Preview: PMG, BMP, [✓ ALL]
Mode: GED +, [✓ DLI], GED --, PGR
Pixel: 1x1, [✓ 2x1], 4x1
GTIA: [✓ 16G], 9C, 16C
Screen: Narrow, [✓ Normal], Wide
Video: [✓ GTIA], VBXE

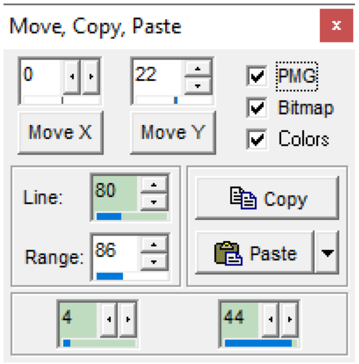
(y: 6) (r: 0) | Charsets: 1 | Mode: DLI | Preview: ALL | Image (GTIA / NORMAL / CHAR): 160 x 240 x 4 (9600 bytes)

G2F trickery to the max: ten colors per line using *Player* objects, multicolor *Players* and the fifth color.

Additional useful functions

Moving a whole picture or parts of it

We talked briefly about the Move/Copy/Paste feature a bit earlier, but only about the copy/paste aspect of it. You can also use it move sections of your picture, or all of it. Open the panel with *Image > Move, Copy, Paste* (shortcut: **Shift+M**).



Line indicates the first line of your desired selection, *Range* the height for it. The two numbers at the bottom of the panel represents the width of the selection; leave it at 4 and 44 for the full width of a 160-pixel wide image. The numbers at the top for *Move X* and *Move Y* represent the offset the selection will move by. Remember you can also use the tick boxes on the right to affect the *Bitmap* or the *Players/Missiles* content specifically, or both.

Exporting a picture as an executable

Your picture is complete and looks amazing in Graph2Font, but how do you get it to display on the actual hardware or an emulator? It's as simple as selecting *File > Save As…* and choosing *XEX Atari Executable (*.xex)* as the file type. You can now drag and drop the *XEX* file directly into an Altirra application window to have it displayed in the emulator.

Most important keyboard shortcuts

In the main window

Ctrl+E	Check limitations (number of changes per line)
Ctrl+Z	Undo
Shift+Ctrl+Z	Redo
Shift+M	Move, copy or paste an image or a part of it including *Bitmaps*, *PMG* and colors
Alt+C	Edit colors of *Bitmap* graphics
Alt+L	Edit palette
Alt+P	Edit *Players/Missiles*
Alt+Z	Zoom mode

In the Zoom window

1…8	Show/hide *Player/Missile* objects
Q	Select previous color in *Bitmap* graphics palette
W	Select next color in *Bitmap* graphics palette
O	Draw pixels in *Player/Missile* graphics
A	Erase pixels in *Player/Missile* graphics

Thank you!

- Od1niec for your massive help and inspiring works.
- Carrion, Piesiu, Ooz, Rocky and Tiger for all the fantastic pixel graphics done on Atari XL/XE.
- The coders of Graph2Font for an awesome art package.
- Agenda, Lamers, New Generation (and others!) for keeping the Atari 8-bit demoscene alive.
- Excellence in Art, whose maxYMiser handbook inspired me to write this document.

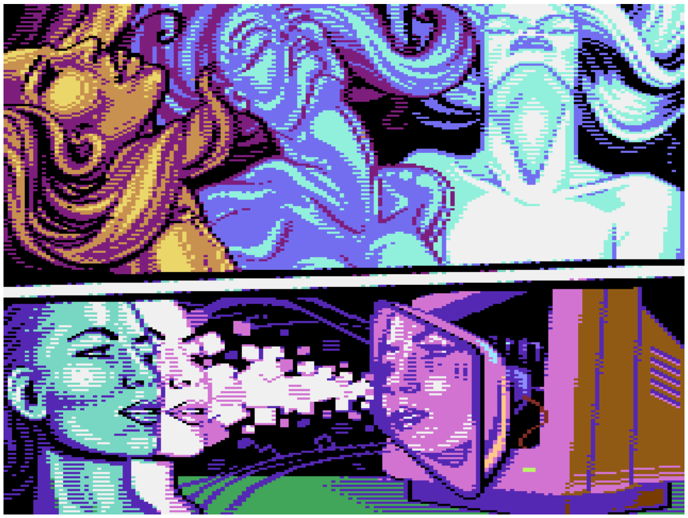
Contacting the author

Are you missing anything in this guide? Do you want to talk about pixel graphics and chip music? Don't hesitate to contact me:

- *modmate[at]smfx.st*
- https://soundcloud.com/modmate

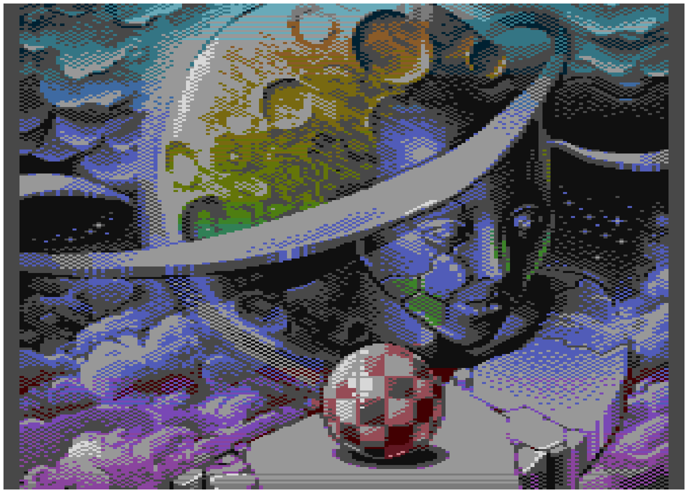
Note from Exocet:

Sadly, mOdmate cannot be contacted anymore but if you spot any mistake or have suggestions, feel free to contact me at *exocet[at]atari.org*. I'll do my best to keep this guide accurate and up to date so it remains as helpful as possible.



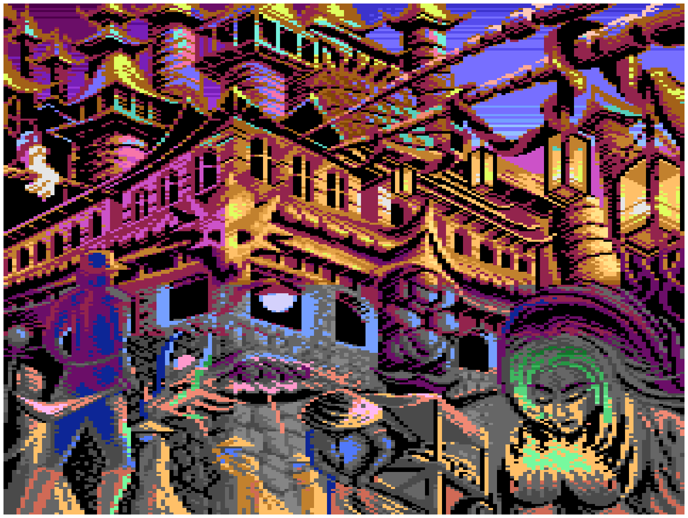
Transmission by Ripek (2014)

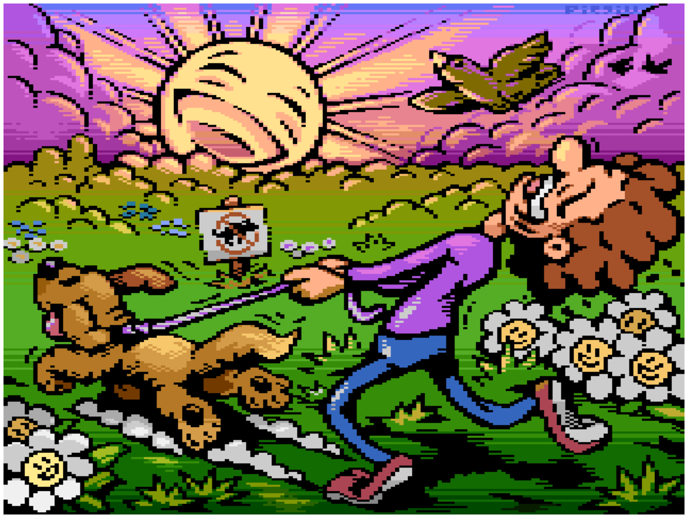
The Gift by Ripek (2016).



YoomBlazer by Carrion (2019)

Sad by Rocky (2020).


Lost In Kyoto by Tiger (2021)


Radość O Poranku by Piesiu (2025)