The bB Titlescreen Kernel 1.5 Documentation

Thanks to the fine folks at atariage.com for all the suggestions and inspiration, Jeff Wierer for his fantastic visualbB tools, and a special thanks to Fred Quimby for creating batari Basic, the Harmony cart, and continually supporting the 2600 dev community.

Mike Saarna (aka RevEng)

Contents

1. What is the Titlescreen Kernel? What can it do?

The Titlescreen Kernel is a custom assembly module that allows you to display a high quality titlescreen in your batari Basic game, without having to write any assembly code yourself.

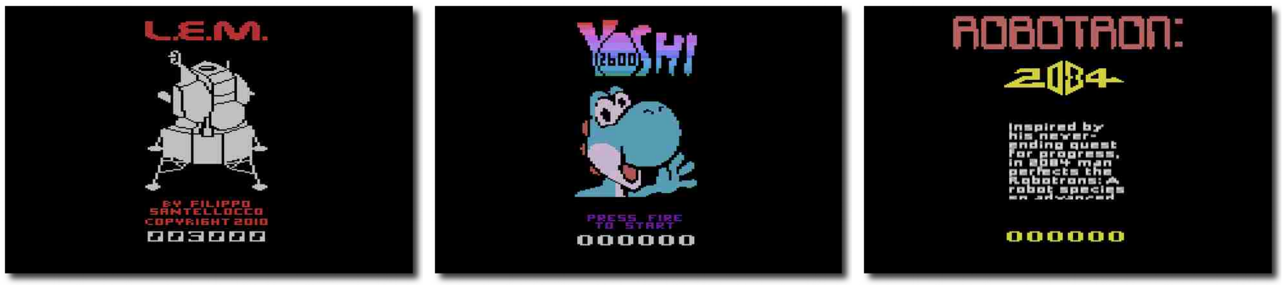


figure 1. Some examples of titlescreens made with the Titlescreen Kernel.

The titlescreen doesn't have to be just a pretty picture; your batari Basic program can manipulate different parts of the titlescreen - scrolling an image, flipping through different frames of an animation, changing colors, etc.

2. Overview: How is a titlescreen made?

A custom titlescreen is made up of multiple display minikernels. Each selected minikernel is stacked on another. You choose which minikernels you want displayed in your titlescreen, and what data they are displaying

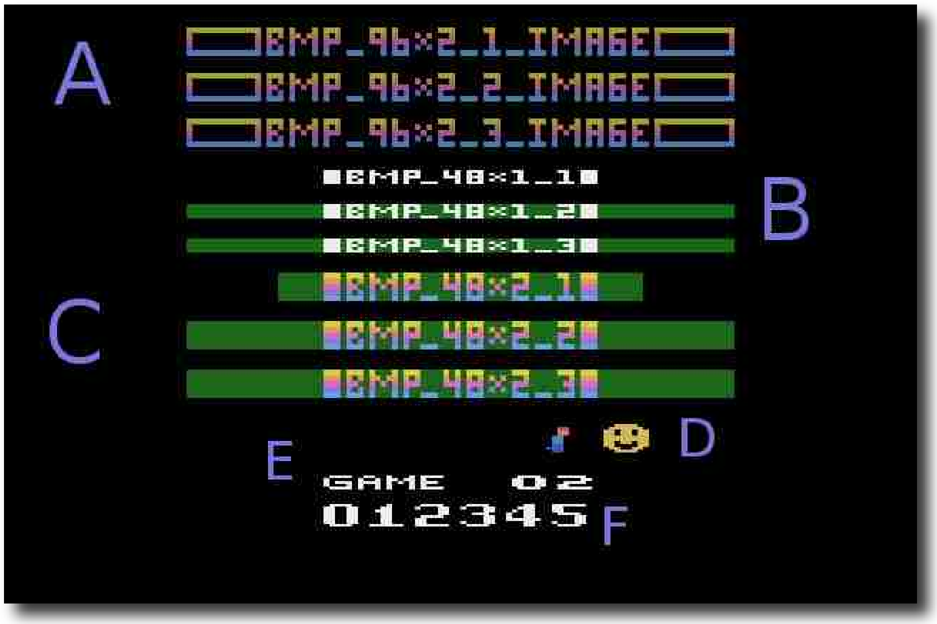


figure 2. All of the minikernels available in the Titlescreen Kernel. You select which of these you want for your titlescreen.

All of the minikernels that display pictures have 3 copies. This is for flexibility in your layout; each copy can display a different picture, be a different color, etc.

The minikernels you can select from are:

A.*the 96x2 minikernel* - this displays a 96-pixel wide bitmap image.

The "x2" part of the name means that this minikernel display pixels using pixels that 2 lines tall on your TV; the pixels it displays will be square, more or less.

With this minikernel you set the color of each line, so it's possible to create a very colorful logo or graphic with it.

To display a bitmap this wide, this minikernel employs a flicker technique called flickerblinds. While flickerblinds is not as objectionable to most people as regular flicker is, it can still be distracting.

B.*the 48x1 minikernel* - this displays a 48-pixel wide bitmap image.

The "x1" part of the name means that this minikernel display pixels using a single line on your TV; the pixels are half height, making them look like rectangles instead of squares.

This is a single color minikernel, though it's possible to set the color dynamically using a variable in your batari Basic program. You can also have a background playfield behind the image of any color.

C.*the 48x2 minikernels* - these allow you to display a 48 wide bitmap image.

The "x2" part of the name means that this minikernel display pixels using 2 lines on your TV.

This means the pixels it displays are square, more or less.

With this minikernel you set the color of each line, so it's possible to create a very colorful logo or graphic with it. You can also have a background playfield behind the image of any color.

D. *The player minikernel* – this allows you to display up to 2 players/sprites within its display area.

E. *the gameselect minikernel* - this allows you to display the word "game" followed by the value of the bB "gamenum" variable.

F. *the score minikernel* - this allows you to display the bB score variable, similar to the bB score display. This is useful for displaying the user's score from the last game.

G. *the space minikernel (not pictured)* - this is used to create blank space between two minikernels.

The data you display with the minikernels must not exceed the number of scanlines available on the screen. (~85 double-lines, or ~170 single-lines). Also, you'll need to take into account that each minikernel you choose will take a few lines to initialize itself.

If you wind up exceeding the scanline count, on real hardware the TV display will roll, and in some emulators NTSC binaries will be detected as PAL.

3. Before we start.

The tools.

Designing a titlescreen is a process that involves creating and manipulating images. You'll need some utlities to do this image work with.

Img2Code

The Atari 2600 doesn't have the horsepower to work with a complex bitmap file format like your computer does. Instead we need to take a bitmap file and convert it into something that bB and the Atari can use, with the Img2Code utility.

Img2Code is part of the most excellent Visual bB IDE, and can be downloaded from:
http://www.atariage.com/forums/topic/123849-visual-bb-1-0-a-new-ide-for-batari-basic/

Graphics Editor

Any graphics editor that can save "bmp" or "png" format files will do. Pick whichever graphics editor you're most comfortable with using.

If you have no preference, The Gimp, is a freely-available fully-featured editor. It can be downloaded from:
http://www.gimp.org/

The techniques.

Even though the Titlescreen Kernel allows you to create graphics at a high resolution for the Atari 2600, we're still talking about images that are 48 or 96 pixels wide. To put that in perspective, many modern desktop computers have *icons* that are larger than that, with a lot more color depth!

It can be a challenge to create an attractive image for your program. One technique I use is creating a simplified drawing at a higher resolution, and then letting my graphics program scale down the image for me, and touch up the final result at the target resolution. (48 pixels wide, or 96 pixels wide)

This was used to good effect for the L.E.M. titlescreen. (L.E.M. is a very good Atari 2600 homebrew by Filippo Santellocco.) A high resolution NASA technical drawing of the lunar module was simplified by drawing over the original image, resizing to a 48 pixel wide format, and following up with some touch ups.

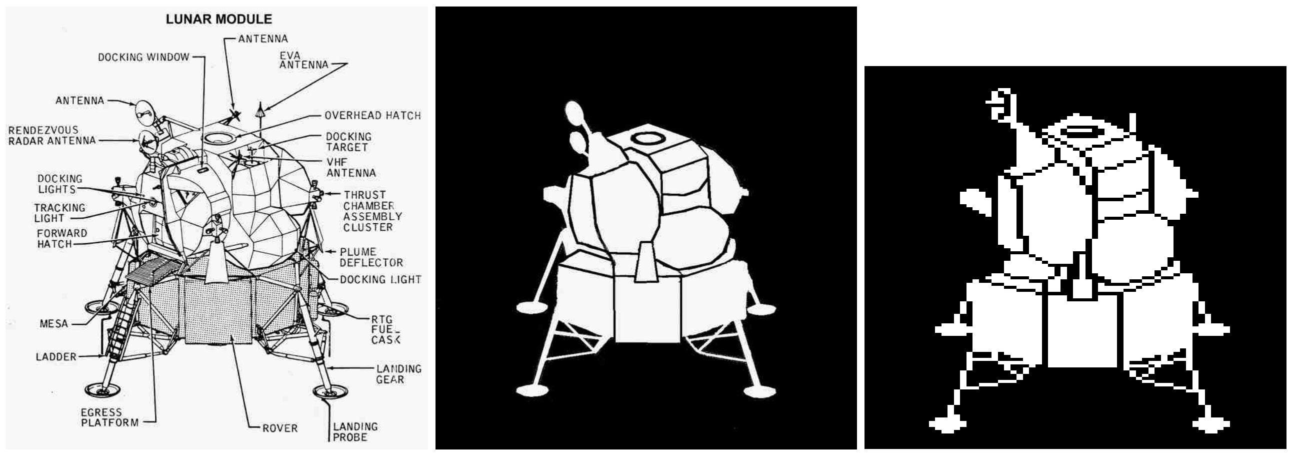
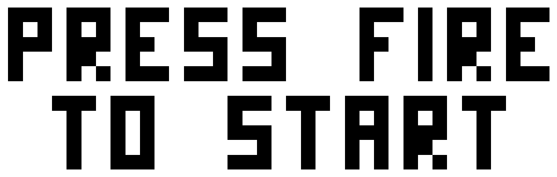


figure 3. the LEM titlescreen graphics in progress.

It's also sometimes useful to start out in the target resolution. This is how all of the small-text graphics were created for the titlescreen examples shown in this document.



A new 48 pixel wide image was created, the image was zoomed to fill the screen, and then the individual pixels were colored.

It's also important that you remove any anti-aliasing from the images you create. The VCS won't be able to show all those nice shades of color, and they'll only wind up as extra unwanted blobs in the image.

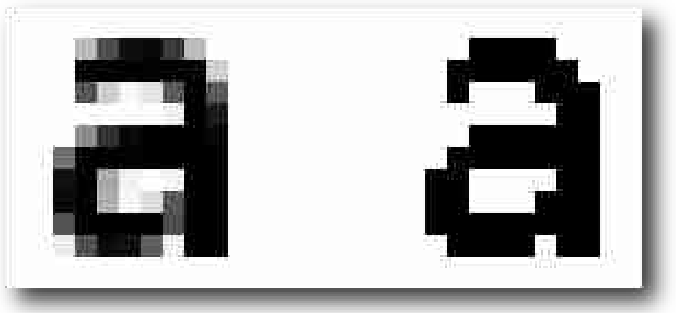


figure 3. The letter "a" with and without antialiasing, magnified.
The 2600 can only handle displaying the one on the right.

4. Example 1 - A simple titlescreen.

Step 1. Creating the graphics

The first part of designing a titlescreen is figuring out what you want to display. Do you want a logo? A graphic? Copyright notice? Anything else?

In this example we'll walk through the creation of a fictional game called "UV Protector". In this example we'll create a titlescreen with a logo, a graphic, a copyright notice, a game number display, and the score.

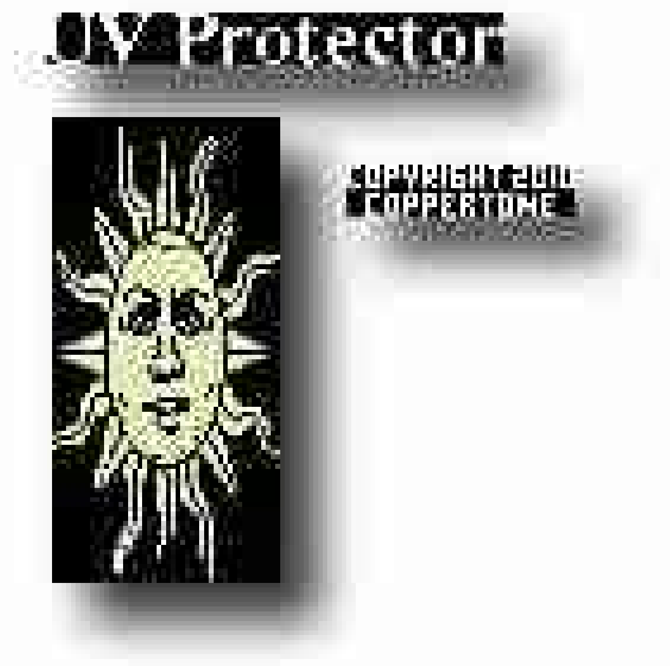


figure 4. First design your images in your graphics software.
You want the width of the images to match the kernel they're going in.

Since "UV Protector" is a rather large name, we designed a 96 pixel wide graphic that we're going to use with the 96x2 minikernel.

The sun graphic was a public domain image taken from a woodcut. The image was resized so it was 48 pixels wide and 96 pixels tall. The plan is to use it with the 48x1 minikernel, which has pixels half as tall as they are wide - this kernel offers the maximum resolution the Atari 2600 can produce.

The copyright notice fits nicely into a 48 wide image. We'll use this with a 48x1 minikernel as well, because we want the notice to be as small as possible.

Note: You should make sure that your images are all monochrome before you move on with the next step. Converting images with many colors can cause problems later with the conversion process.

Step 2. Placing the Titlescreen Kernel in your game's project directory

When you download and unzip the Titlescreen Kernel, you'll see it has a "titlescreen" directory, with a number of files in it.

```
titlescreen:
  48x1_1_image.asm  48x2_3_image.asm  gameselect_image.asm
  48x1_2_image.asm  96x2_1_image.asm  score_image.asm
  48x1_3_image.asm  96x2_2_image.asm  titlescreen_color.asm
  48x2_1_image.asm  96x2_3_image.asm  titlescreen_layout.asm
  48x2_2_image.asm  asm
```

Simply copy this directory over to your game project directory. We'll customize the files inside it in the next step.

Step 3. Importing the graphics into the titlescreen kernel

To convert our logo, we're going to perform the following steps:

i. Run the IMG2CODE program.
ii. Choose the "File+Open" menu and open the bitmap file.
iii. Ensure the following options are set in IMG2CODE.

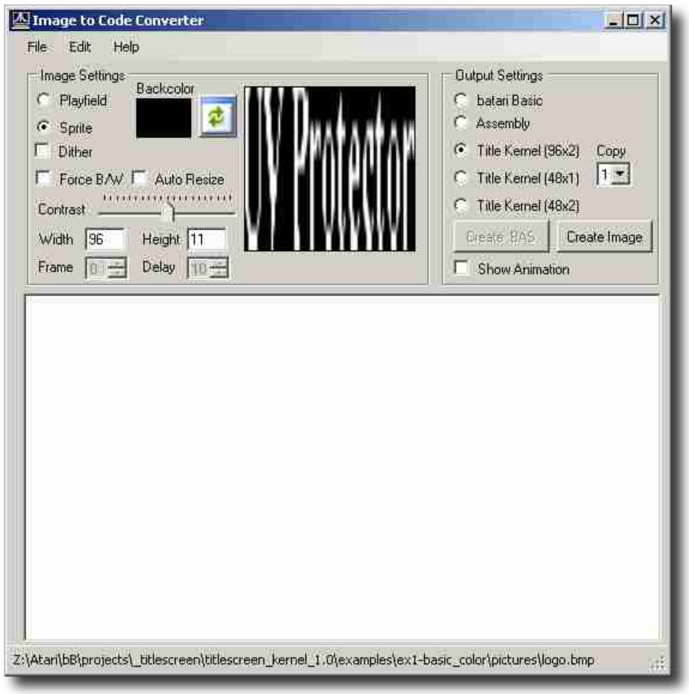


Image Settings: Playfield [✓ Sprite]; Dither: unchecked; Force B/W: unchecked; Auto Resize: unchecked
Width: 96 Height: 11
Output Settings: batari Basic, Assembly, [✓ Title Kernel (96x2)], Title Kernel (48x1), Title Kernel (48x2); Copy: 1
Show Animation: unchecked

Z:\Atari\bB\projects_titlescreen\titlescreen_kernel_1.0\examples\ex1-basic_color\pictures\logo.bmp *figure 5. If you need to change the "backcolor" box, just move your mouse over the image and select part of the background.*

iv.Click on the "Create Image" button.

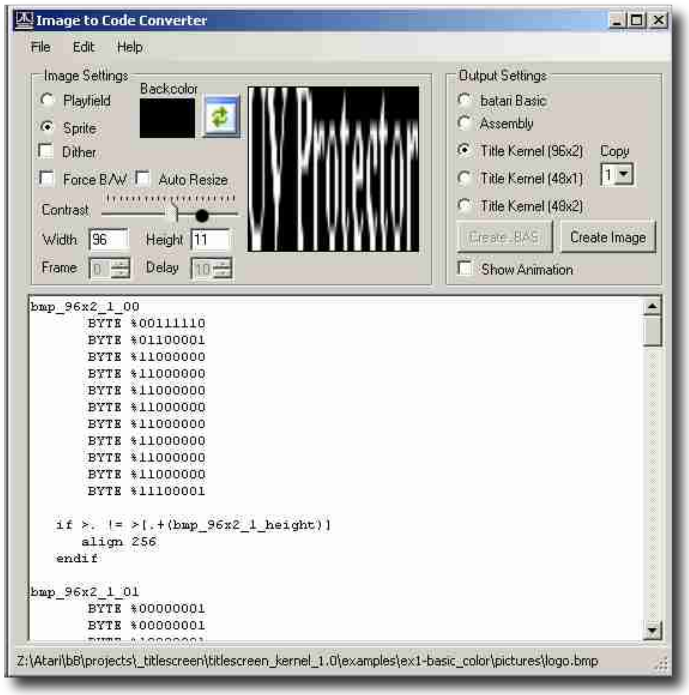


Image Settings: Playfield ( ), [✓ Sprite], Dither [ ], Force B/W [ ], Auto Resize [ ], Width 96, Height 11, Frame 0, Delay 10

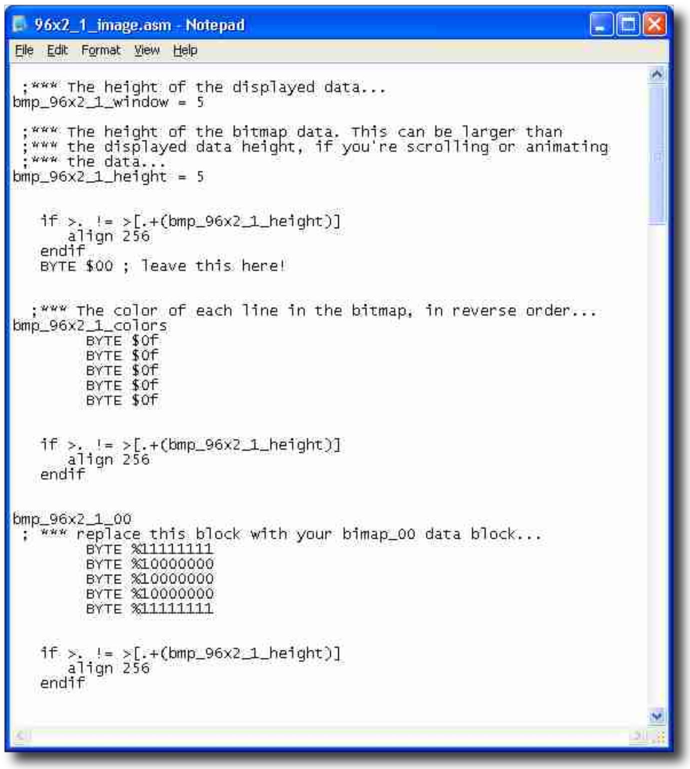
Output Settings: batari Basic ( ), Assembly ( ), [✓ Title Kernel (96x2)], Title Kernel (48x1) ( ), Title Kernel (48x2) ( ), Copy 1

```
bmp_96x2_1_00
        BYTE %00111110
        BYTE %01100001
        BYTE %11000000
        BYTE %11000000
        BYTE %11000000
        BYTE %11000000
        BYTE %11000000
        BYTE %11000000
        BYTE %11000000
        BYTE %11000000
        BYTE %11100001

    if >. != >[.+(bmp_96x2_1_height)]
       align 256
    endif

bmp_96x2_1_01
        BYTE %00000001
        BYTE %00000001
```

Z:\Atari\bB\projects_titlescreen\titlescreen_kernel_1.0\examples\ex1-basic_color\pictures\logo.bmp

figure 6. The text area is filled with the code we need to import into bB.

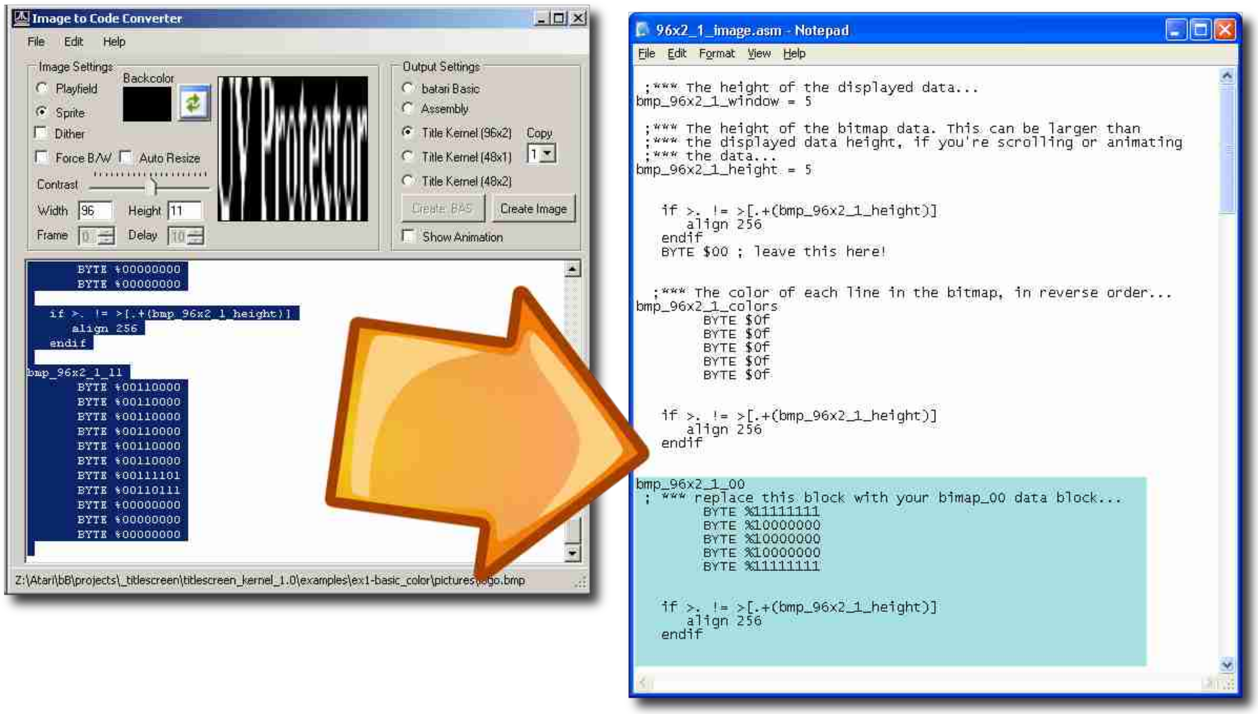
v. Using notepad, open the "96x2_1_image.asm" file located in the titlescreen directory.



```
 ;*** The height of the displayed data...
bmp_96x2_1_window = 5

 ;*** The height of the bitmap data. This can be larger than
 ;*** the displayed data height, if you're scrolling or animating
 ;*** the data...
bmp_96x2_1_height = 5

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif
   BYTE $00 ; leave this here!

  ;*** The color of each line in the bitmap, in reverse order...
bmp_96x2_1_colors
        BYTE $0f
        BYTE $0f
        BYTE $0f
        BYTE $0f
        BYTE $0f

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif

bmp_96x2_1_00
 ; *** replace this block with your bimap_00 data block...
        BYTE %11111111
        BYTE %10000000
        BYTE %10000000
        BYTE %10000000
        BYTE %11111111

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif
```

*figure 7. the 96x2_**1**_image.asm file contains the image data for the **1st** copy of the 96x2 minikernel.*

vi.Highlight the entire text area within IMG2CODE and copy+paste it over everything starting from the bmp_96x2_1_00 label to the end of file.



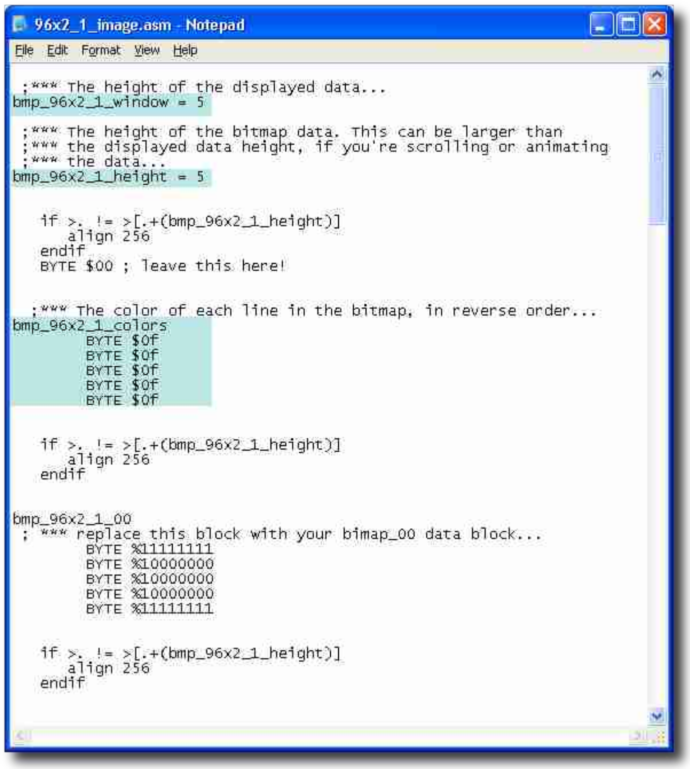
```
        BYTE %00000000
        BYTE %00000000

  if >. != >[.+(bmp_96x2_1_height)]
     align 256
  endif

bmp_96x2_1_11
        BYTE %00110000
        BYTE %00110000
        BYTE %00110000
        BYTE %00110000
        BYTE %00110000
        BYTE %00110000
        BYTE %00111101
        BYTE %00110111
        BYTE %00000000
        BYTE %00000000
        BYTE %00000000
```

```
 ;*** The height of the displayed data...
bmp_96x2_1_window = 5

 ;*** The height of the bitmap data. This can be larger than
 ;*** the displayed data height, if you're scrolling or animating
 ;*** the data...
bmp_96x2_1_height = 5

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif
   BYTE $00 ; leave this here!

  ;*** The color of each line in the bitmap, in reverse order...
bmp_96x2_1_colors
         BYTE $0f
         BYTE $0f
         BYTE $0f
         BYTE $0f
         BYTE $0f

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif

bmp_96x2_1_00
 ; *** replace this block with your bimap_00 data block...
         BYTE %11111111
         BYTE %10000000
         BYTE %10000000
         BYTE %10000000
         BYTE %11111111

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif
```

figure 8. Copy and paste the data. Be careful not to disturb any of the other sections.

vii. In notepad, adjust the bmp_96x2_window and bmp_96x2_height values so they equal your image height.

The bmp_96x2_colors list should also be adjusted, so it has a color entry for each row in your picture, and the list should be in reverse order, with the color for the bottom of your image being first.



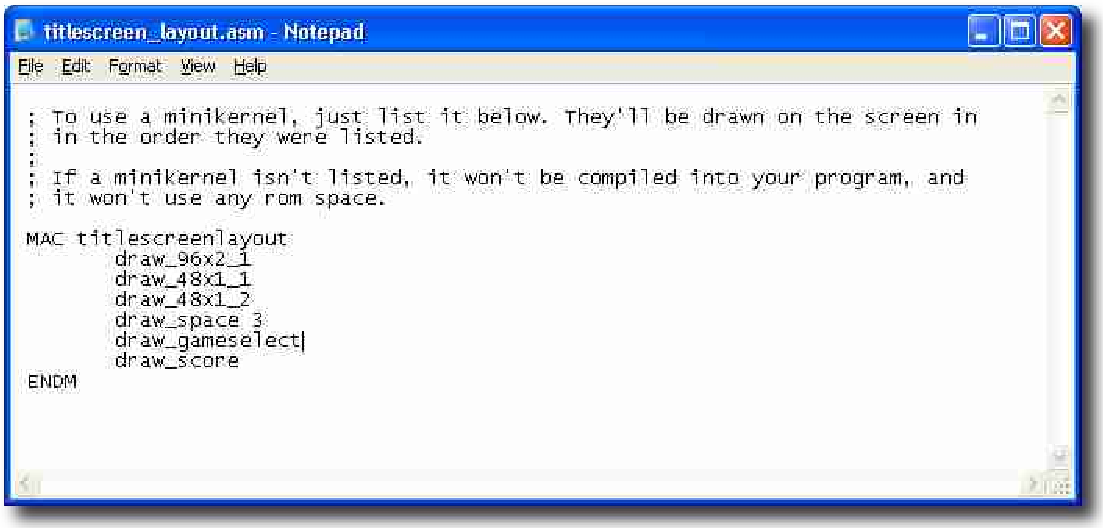
```
 ;*** The height of the displayed data...
bmp_96x2_1_window = 5

 ;*** The height of the bitmap data. This can be larger than
 ;*** the displayed data height, if you're scrolling or animating
 ;*** the data...
bmp_96x2_1_height = 5

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif
   BYTE $00 ; leave this here!

  ;*** The color of each line in the bitmap, in reverse order...
bmp_96x2_1_colors
         BYTE $0f
         BYTE $0f
         BYTE $0f
         BYTE $0f
         BYTE $0f

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif

bmp_96x2_1_00
 ; *** replace this block with your bimap_00 data block...
         BYTE %11111111
         BYTE %10000000
         BYTE %10000000
         BYTE %10000000
         BYTE %11111111

   if >. != >[.+(bmp_96x2_1_height)]
      align 256
   endif
```

figure 9. The remaining values to adjust.

viii. Save the file.

ix. Repeat these steps for the other 2 graphic files. They are both 48x1 type images, so you should edit the bmp_48x1_1_image.asm and bmp_48x1_2_image.asm files. When working with the image for the bmp_48x1_2_image.asm file, be sure to set the "Copy" control in Img2Code to "2".

x. The last assembly file we need to work with is the titlescreen_layout.asm file. Edit it in notepad and make sure it lists all of the minikernels we worked on...



```
; To use a minikernel, just list it below. They'll be drawn on the screen in
; in the order they were listed.
;
; If a minikernel isn't listed, it won't be compiled into your program, and
; it won't use any rom space.

MAC titlescreenlayout
        draw_96x2_1
        draw_48x1_1
        draw_48x1_2
        draw_space 3
        draw_gameselect
        draw_score
ENDM
```

figure 10. The layout file. It determines which minikernels are displayed, and the order they are displayed in.

We also included the game selection minikernel, so the user can pick a game number, and the score minikernel, so we can display the score variable.

It's also worth mentioning that the layout file distributed with the Titlescreen Kernel has a commented list of all of the available minikernel names, so you don't need to memorize all of their names.

Step 4. Modifying your bB program to use the titlescreen

In your bB program, 2 additions need to be made to use the titlescreen.

1. You need to add an "include" statement in the empty bank where you want to store your titlescreen. In this example we use bank 2

```
 bank 2
 asm
 include "titlescreen/asm/titlescreen.asm"
end
```

2. And you need to call the titlescreen drawing routine in a loop, just like you'd use the bB drawscreen command

```
titlepage
 gosub titledrawscreen bank2
 if joy0fire || switchreset then goto gamestart
 goto titlepage
```

Then just compile your bB program as usual and run.

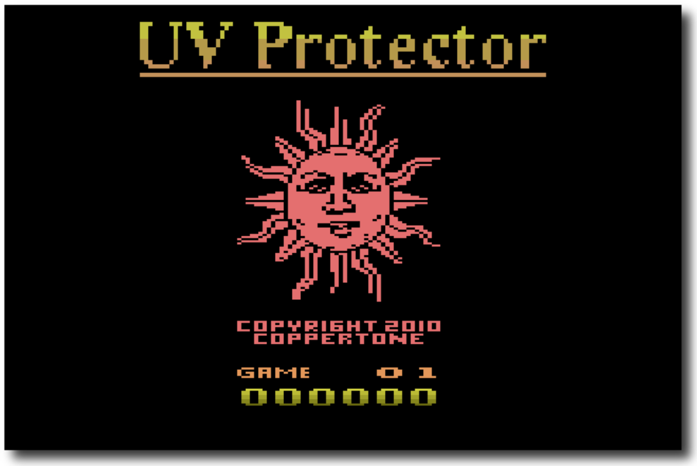
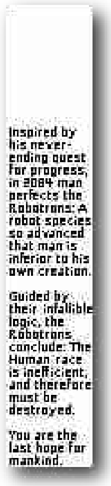


figure 11. The final product,
creepy sun and all!

5. Example 2 - A titlescreen with a scrolling graphic.

We're now going to assume that you've learned how to import data into the bitmap minikernels from the first example, so this example will just focus on what needs to be done differently to add the scroller.

We start by making a graphic of the text we want to scroll. We'll make it a 48x256 image, since 256 lines is the maximum height we can use, and we want as much text as possible.



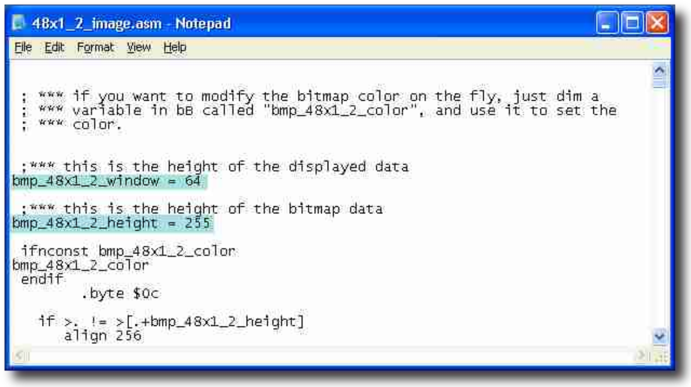
Inspired by
his never-
ending quest
for progress,
in 2084 man
perfects the
Robotrons: A
robot species
so advanced
that man is
inferior to his
own creation.

Guided by
their infallible
logic, the
Robotrons
conclude: The
Human race
is inefficient,
and therefore
must be
destroyed.

You are the
last hope for
mankind.

figure 12. The blank area at the top is so the text starts
out blank and eventually scrolls up.

We run the image through IMG2CODE and import the data blocks into the minikernel just as before. But this time in the assembly file we'll set the height to 256, and the window to 64.



```
 ; *** if you want to modify the bitmap color on the fly, just dim a
 ; *** variable in bB called "bmp_48x1_2_color", and use it to set the
 ; *** color.

 ;*** this is the height of the displayed data
bmp_48x1_2_window = 64

 ;*** this is the height of the bitmap data
bmp_48x1_2_height = 255

 ifnconst bmp_48x1_2_color
bmp_48x1_2_color
 endif
        .byte $0c

   if >. != >[.+bmp_48x1_2_height]
      align 256
```

figure 13. We're only going to display 64 lines of the image at a time.

To define which minikernels are used in the titlescreen, edit the titlescreen_layout.asm file so it reads as follows:

```
MAC titlescreenlayout
     draw_96x2_1
     draw_48x1_1
     draw_space 10
     draw_48x1_2
     draw_space 20
     draw_score
ENDM
```

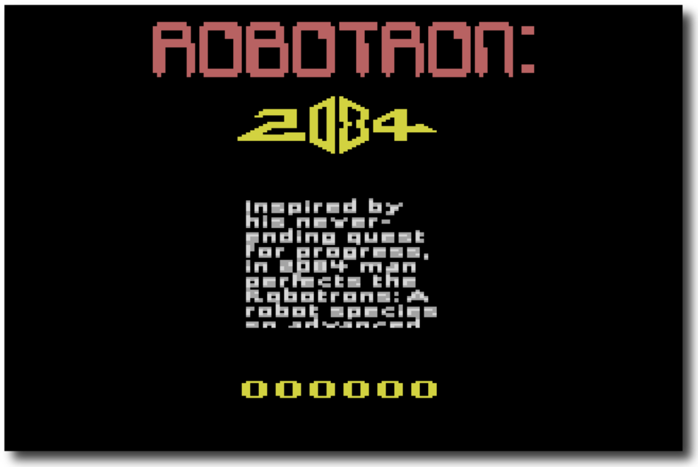
To add scrolling, you need to dim an image index variable in your bB code. Since we used the second copy of the 48x1 kernel for our text, this variable would be named bmp_48x1_**2**_index.

```
dim bmp_48x1_2_index=a
```

When bmp_48x1_2_index=0, the top of the image is shown. If we gradually increase bmp_48x1_2_index, it will scroll though the image. The following code will increase the index every 8th frame (so we don't scroll too fast) and then stop when the index is greater than 191.

```
titlepage
 gosub titledrawscreen bank2
 frame=frame+1
 temp1=frame&%00000111
 if temp1=0 && bmp_48x1_2_index<191 then bmp_48x1_2_index=bmp_48x1_2_index+1
 if joy0fire || switchreset then goto gamestart
 goto titlepage
```

After compiling the bB program, we have our scolling text titlescreen...


ROBOTRON:

2084

Inspired by
his never-
ending quest
for progress,
in 2084 man
perfects the
Robotrons: A
robot species

000000

figure 14. Man... he's always getting inspired by his never-ending quest for progress.

6. Example 3 - A titlescreen with an animated graphic.

In principal an animated titlescreen is no different than a scrolling titlescreen - you just move the index in larger jumps.

First we prepare an image with each frame of the animation stacked on top of the other.

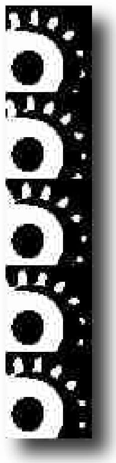


figure 15. 5 frames of a gear-in-the-shadows animation. Each frame is 48 lines tall.

Import the image into the 48x2 minikernel, just like we did with the previous scrolling example.

To define which minikernels are used in the titlescreen, edit the titlescreen_layout.asm file so it reads as follows:

```
MAC titlescreenlayout
     draw_48x1_2
     draw_space 10
     draw_48x2_1
     draw_space 20
     draw_48x1_3
```

```
        ENDM
```

Then just add some code to jump the index from frame to frame.

```
        bmp_48x2_1_index=0

      titlepage
        gosub titledrawscreen bank2
        frame=frame+1
        temp1=frame&%00000111
        if temp1=0 then bmp_48x2_1_index=bmp_48x2_1_index+48:if bmp_48x2_1_index=240
then bmp_48x2_1_index=0
        if joy0fire || switchreset then goto gamestart
        goto titlepage
```

Then after compiling...

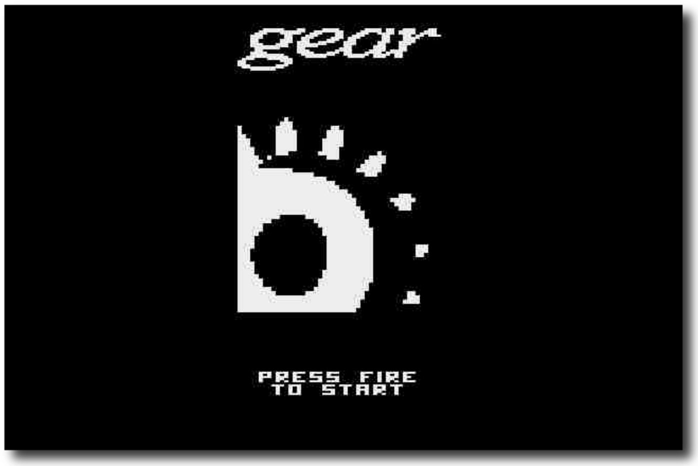


figure 16. the gears are animated.
You'll need to trust me on that one.

7. Example 4 - A titlescreen with flickered color.

This time our goal is going to be to create a color graphic by displaying an animation with 2 frames. When the first frame is up we'll display one color, and when the second frame is up we'll display another color.

Keeping color mixing in mind, we created the following concept graphic

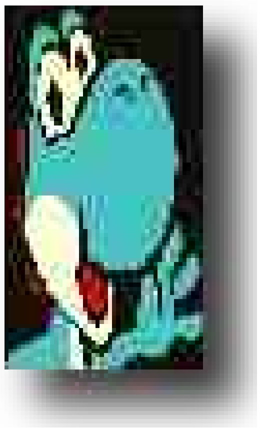


figure 17. Our new character, Mr. Copyright Dinosaur

We picked the colors cyan and red, because they mix together to make white, more or less.

We need to manually create the two animation frames. The top frame needs to have any picture element that should be red or white, and the bottom frame should have any picture element that should be cyan or white.

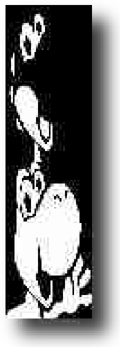


figure 18. We did this manually, by stacking 2 copies of the color dinosaur image and eliminating parts that shouldn't show up in each frame.

Import the images into the respective files as we did in the previous examples.

To define which minikernels are used in the titlescreen, edit the titlescreen_layout.asm file so it reads as follows:

```
  MAC titlescreenlayout
        draw_48x2_1
        draw_48x1_1
        draw_space 6
        draw_48x1_2
        draw_score
  ENDM
```

We then add the code to the bB program that moves the index and changes the color depending on which frame needs to currently be displayed.

```
  titlepage
   frame=frame+1
   bmp_48x1_1_color=$4b :bmp_48x1_1_index=0
   if frame{0} then bmp_48x1_1_color=$ac : bmp_48x1_1_index=87
   gosub titledrawscreen bank2
   if joy0fire then goto gamestart
   goto titlepage
```

Then after compiling...

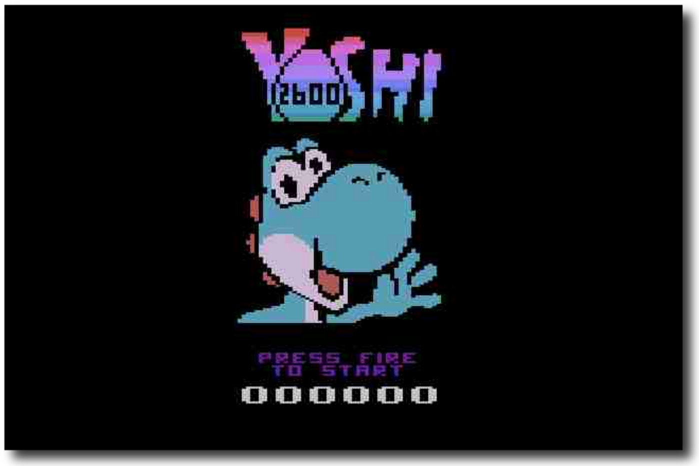


*figure 19. The dinosaur isn't winking at you.
He's flickering.*

8. Example 5 - A titlescreen with player graphics and playfield background

As with the other minikernels, the best place to begin with the player minikernel is to start creating bitmap graphics. If your player has multiple animation frames, stack them one on top of the other.

In this example we'll use a running man and a face...

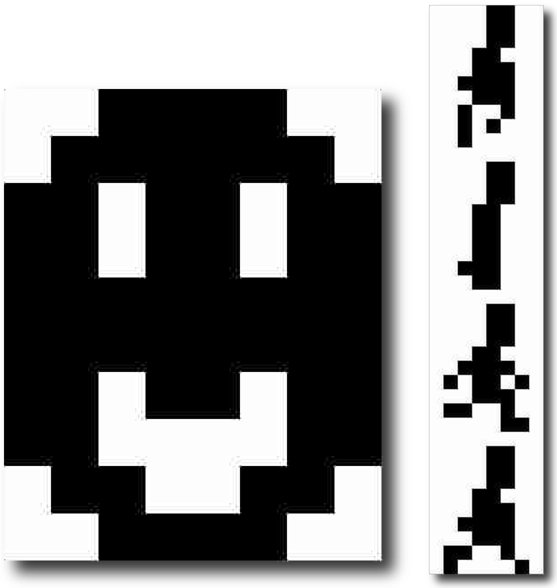


figure 20. the giant pixel face will haunt you in your dreams.

We convert the running man and face using img2code – make note of the settings, as they're different than when we convert data for other minikernels - and then paste the output into the titlscreen/player_image.asm file, under the bmp_player0 and bmp_player1 sections...

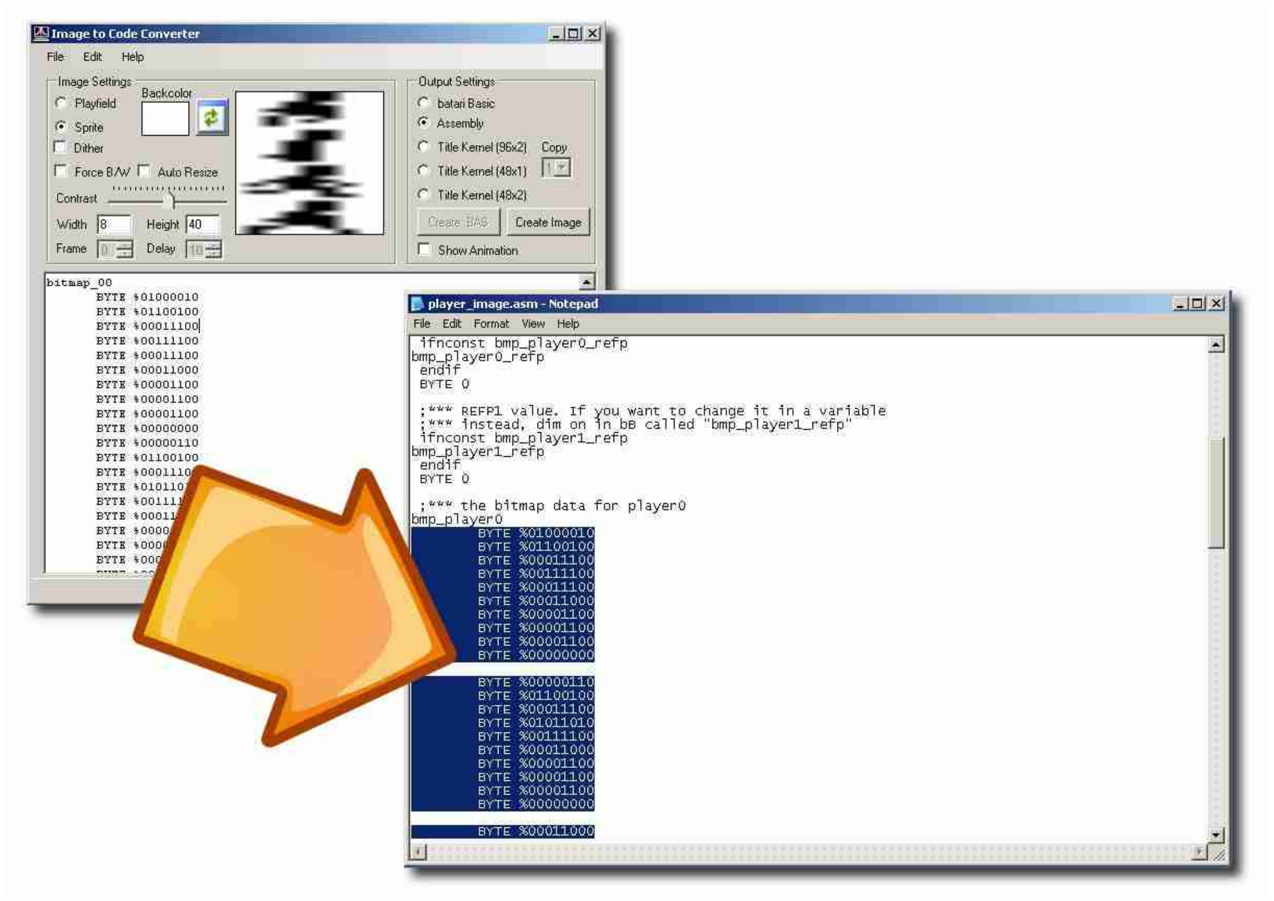


Image Settings: ( ) Playfield [✓ Sprite] ☐ Dither ☐ Force B/W ☐ Auto Resize Backcolor Contrast Width 8 Height 40 Frame 0 Delay 10

Output Settings: ( ) batari Basic [✓ Assembly] ( ) Title Kernel (96x2) ( ) Title Kernel (48x1) ( ) Title Kernel (48x2) Copy 1 ☐ Show Animation

```
bitmap_00
        BYTE %01000010
        BYTE %01100100
        BYTE %00011100
        BYTE %00111100
        BYTE %00011100
        BYTE %00011000
        BYTE %00001100
        BYTE %00001100
        BYTE %00001100
        BYTE %00000000
        BYTE %00000110
        BYTE %01100100
        BYTE %0001110
        BYTE %010110
        BYTE %00111
        BYTE %00011
        BYTE %0000
        BYTE %000
        BYTE %000
```

```
 ifnconst bmp_player0_refp
bmp_player0_refp
 endif
 BYTE 0

 ;*** REFP1 value. If you want to change it in a variable
 ;*** instead, dim on in bB called "bmp_player1_refp"
 ifnconst bmp_player1_refp
bmp_player1_refp
 endif
 BYTE 0

 ;*** the bitmap data for player0
bmp_player0
        BYTE %01000010
        BYTE %01100100
        BYTE %00011100
        BYTE %00111100
        BYTE %00011100
        BYTE %00011000
        BYTE %00001100
        BYTE %00001100
        BYTE %00001100
        BYTE %00000000

        BYTE %00000110
        BYTE %01100100
        BYTE %00011100
        BYTE %01011010
        BYTE %00111100
        BYTE %00011000
        BYTE %00001100
        BYTE %00001100
        BYTE %00001100
        BYTE %00000000

        BYTE %00011000
```

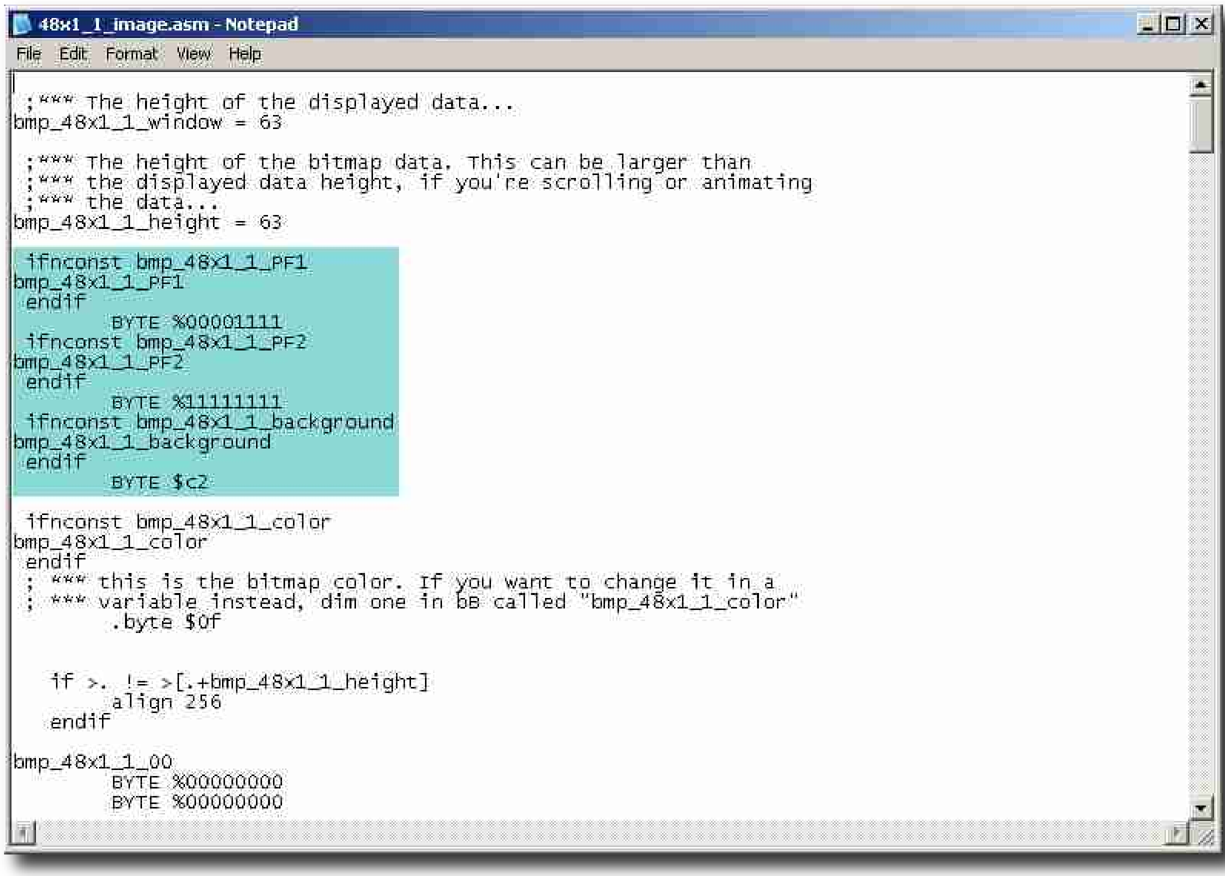
figure 21. copy and paste just the BYTE lines, not the bitmap_00 label at the top.

Looking through the rest of the file, manually fill out these sections:

1. bmp_player_window – this determines the height of the entire player minikernel
2. bmp_player_kernellines – this determines if the height of each player pixel is 1 scanline, 2 scanlines, etc.
3. bmp_player0_height, bmp_player1_height – the actual height of any one frame of the player.
4. bmp_color_player0, bmp_color_player1 – the color list for each player sprite. If your players have multiple frame data, like the running man example does, you should use multiple frames of the color data also.

In this example we'll also demonstrate playfield backgrounds by adding a blue background behind the game logo in this demonstration.

First open the bitmap file and paste the image data into 48x1_1_image.asm from img2code, as we did in the previous examples. Then enter “BYTE %00001111” under the entry for bmp_48x1_1_PF1. Similarly, enter “BYTE %11111111” under the entry for bmp_48x1_1_PF2. This will create a playfield background box in the middle of the minikernel. Lastly, we define the color of this box by changing the value under the 48x1_1_background entry.|



```
 ;*** The height of the displayed data...
bmp_48x1_1_window = 63

 ;*** The height of the bitmap data. This can be larger than
 ;*** the displayed data height, if you're scrolling or animating
 ;*** the data...
bmp_48x1_1_height = 63

 ifnconst bmp_48x1_1_PF1
bmp_48x1_1_PF1
 endif
        BYTE %00001111
 ifnconst bmp_48x1_1_PF2
bmp_48x1_1_PF2
 endif
        BYTE %11111111
 ifnconst bmp_48x1_1_background
bmp_48x1_1_background
 endif
        BYTE $c2

 ifnconst bmp_48x1_1_color
bmp_48x1_1_color
 endif
 ; *** this is the bitmap color. If you want to change it in a
 ; *** variable instead, dim one in bB called "bmp_48x1_1_color"
        .byte $0f


   if >. != >[.+bmp_48x1_1_height]
        align 256
   endif

bmp_48x1_1_00
        BYTE %00000000
        BYTE %00000000
```

figure 22. you need to edit the 3 lines that begin with "BYTE"

To define which minikernels are used in the titlescreen, edit the titlescreen_layout.asm file so it reads as follows:

```
MAC titlescreenlayout
     draw_48x1_1
     draw_player
     draw_space 25
     draw_48x1_2
     draw_score
ENDM
```

Now we're ready to update our basic file, by including the titlescreen.asm like we did with the previous examples.

Moving the players around in the player minikernel is just a matter of changing player0x, player0y, player1x, and player1y. If you want to hide one of your players, just change their y coordinate so it's not displayed.

To change animation frames for our running man, we also need to dim an index variable for player0 – bmp_player0_index. Changing this variable will change the section of sprite data used for the display. For a sprite with a height of 10, setting the index to 0, 10, 20, ..., will step through the different animation frames.

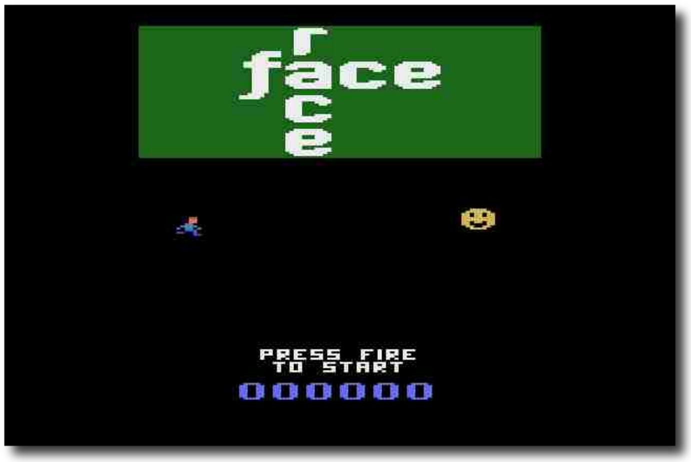


figure 23. is it race-face or face-race?

9. Details and Troubleshooting

Q: My colors are all weird. What caused this?

You're probably using too many screen lines. The Titlescreen Kernel can display ~85 lines using 2-line minikernels, or about ~170 lines using 1-line minikernels.

Also, sometimes dasm seems to wrongly indentify the color labels in the minikernel image files, when code shifts in one of the assembler passes. Make sure that the Titlescreen Kernel is the first thing in the bank to avoid this.

Q: I'm running out of room in the bank. What can I do?

- Make sure that the minikernel has the bank fully to itself.
- Use smaller images and more "space" between them.
- The 48x1 and 48x2 minikernels actually wind up using the same core minikernel code, so if you build a titlescreen around these you will minimize the amount of rom the titlescreen code takes up.