001 – Display a background

Prerequisite : 000 – Create a new game project

This tutorial describes how to display :

- a solid color background
- a background image

Introduction

As explained in the 000 tutorial, a game is made of one or several game mode(s).

A game mode is a program loaded in ram and running an infinite loop.

This tutorial will introduce another concept of the game engine : an act.

- An act is an initialization routine for a game mode
- A game mode may contain zero or more acts

The game engine provide some facilities to load an act, you will be able to set :

- a palette
- a screen border
- a solid color background
- an background image

improvement load a tile map, load objects

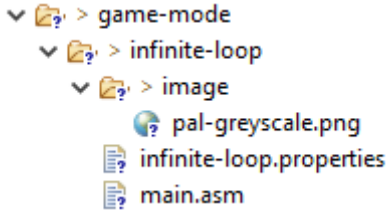
Solid color background

File structure

Enter the infinite-loop directory and create a directory named : **image**

Put this file in the image directory :

pal-greyscale.png



This file will be used to load a color palette.

All images processed by the game engine must be **8 bits indexed colors** png.

Color index 0 is used to define transparent pixels.

Color indexes 1-16 are used to define color pixels.

Game Mode definition

infinite-loop.properties

To define a game mode act, put this configuration into **infinite-loop.properties** :

```
# Palettes
palette.Pal_greyscale=./game-mode/infinite-loop/image/pal-greyscale.png

# Acts definition
actBoot=solidcolor

act.solidcolor.palette=Pal_greyscale
act.solidcolor.screenBorder=15
act.solidcolor.backgroundSolid=14
```

actBoot property defines the default act to load when a game mode is loaded.

screenBorder and **backgroundSolid** properties reference a color in the 0-15 range mapped in the 1-16 color index of the png file.

Game Mode source code

To load the default act of a game mode, put this code into **main.asm** :

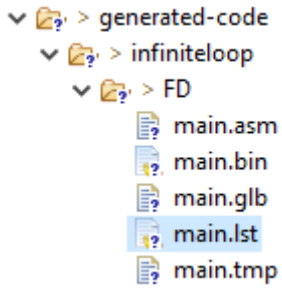
```
        org   $6100
        jsr   LoadAct

* ===========================================================================
* Main Loop
* ===========================================================================
LevelMainLoop
        jsr   WaitVBL
        jsr   UpdatePalette
        bra   LevelMainLoop

* ===========================================================================
* Routines
* ===========================================================================
        INCLUDE "./Engine/Graphics/WaitVBL.asm"
        INCLUDE "./Engine/Palette/UpdatePalette.asm"
        INCLUDE "./Engine/Ram/ClearDataMemory.asm"
```

The call to **LoadAct** will load the act defined in the **actBoot** property.

If no **actBoot** is defined, the LoadAct will be an empty routine (just a **rts**).

Remember :

- **LoadAct** is a generated routine from the Builder, it's not mandatory to use it.
- You are allowed to write a similar routine that will access the resources defined in all the acts definitions, and not just the one declared in **actBoot**.
- If you want to know what's inside **LoadAct**, just have a look to the generated code directory and open **main.lst** file. However this subject will be covered in following tutorials.



Back to the main loop, the program will now swap the video page 2 and 3 for double buffering display (**WaitVBL**). The swap will be executed in sync with screen refresh.

Next, the game engine will check if any color must be changed to the palette.

It's important to set the palette next to the **WaitVBL** call, to ensure no artifact will be displayed at screen. Changing palette colors while spot is on screen will result in visible black dots.

This is why palette change is centralized next to **WaitVBL**.

You are ready to build and run your program.

Details in 000 tutorial

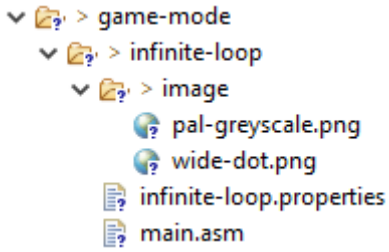
The result :

Background image

File structure

Put this file in the image directory :

wide-dot.png



This file will be used to load a color palette and a background image.

The image size should be 160x200 and must follow color organization explained previously.

Game Mode definition

infinite-loop.properties

To define a new game mode act, put this configuration into **infinite-loop.properties** :

```
# Palettes
palette.Pal_widedot=./game-mode/infinite-loop/image/wide-dot.png

# Acts definition
actBoot=bckimage
```

```
act.bckimage.palette=Pal_widedot
act.bckimage.screenBorder=13
act.bckimage.backgroundImage=./game-mode/infinite-loop/image/wide-dot.png
```

Game Mode source code

To load the default act of a game mode, put this code into **main.asm** :

```
        INCLUDE "./Engine/Macros.asm"
        org   $6100
        jsr   LoadAct

* ===========================================================================
* Main Loop
* ===========================================================================
LevelMainLoop
        jsr   WaitVBL
        jsr   UpdatePalette
        bra   LevelMainLoop

* ===========================================================================
* Routines
* ===========================================================================
        INCLUDE "./Engine/Ram/BankSwitch.asm"
        INCLUDE "./Engine/Graphics/WaitVBL.asm"
        INCLUDE "./Engine/Palette/UpdatePalette.asm"
        INCLUDE "./Engine/Graphics/DrawFullscreenImage.asm"
```

You should have noticed that **ClearDataMemory.asm** have been replaced by **DrawFullscreenImage.asm** since we are now about to display an image instead of a solid color background.

There are also two new includes :

- INCLUDE "./Engine/Macros.asm"
- INCLUDE "./Engine/Ram/BankSwitch.asm"

For the first tutorials, all code were executed from the RAM page 0 ($6100-$9FFF).

Pages 2 and 3 were used as video pages in the $A000-$DFFF range.

Now that we want to display an image, the data must be available somewhere in RAM or ROM.

The game mode loading routine will load the image data in RAM (in case of Disk) or those data will be available in ROM (in case of T.2 if we don't set the RAM flag, more on that later).

To be able to access the data, we need a bank switch routine. For a better performance a macro was designed, this macro will compile as a RAM page switch for the disk version and as a RAM/ROM page switch for the T.2 version.

The bank switch process will be described in following tutorials, you just have to know that the **LoadAct** routine need this functionality to be able to display an image.

You are ready to build and run your program.

Details in 000 tutorial

The result :