HAM Tutorial :: Day 8 :: Sprites #2 - Animation

Well, we covered a lot the first week but, of course, there is so much more to learn. The next thing we'll cover is animating a sprite. What we'll do is create one image that is actually made up of a bunch of smaller images. (Click here to see what I mean.) Then we'll load in each respective piece as the plane flies around the screen.

I downloaded the free graphic off the Internet and then used Photoshop to manipulate it and to create the final image. I decided to make the sprite 64x64 pixels. Since there are 8 different angles, the final image is 64x512 pixels.

Today's example will again be created using a tile mode and there is only one background. Hopefully you all know what command to use to convert the image, but just in case:
gfx2gba -fsrc -m -pbackground.pal -t8 water.bmp

Again, the sprite will not be in a tile/map format, so type:
gfx2gba -D -fsrc -pobject.pal -t8 red_plane_anim.bmp

Your **gfx** directory should have the following files:
background.pal.c
object.pal.c
red_plane_anim.raw.c
water.map.c
water.raw.c

On to the code:

```
// The Main HAM Library
#include <mygba.h>

// Graphics Includes
// gfx2gba -fsrc -m -pbackground.pal -t8 water.bmp
#include "gfx/background.pal.c"
#include "gfx/object.pal.c"
#include "gfx/water.raw.c"
#include "gfx/water.map.c"
// gfx2gba -D -fsrc -pobject.pal -t8 red_plane_anim.bmp
#include "gfx/red_plane_anim.raw.c"

// Defines
#define ANIM_UP        0
#define ANIM_UPRIGHT   1
#define ANIM_RIGHT     2
#define ANIM_DOWNRIGHT 3
#define ANIM_DOWN      4
#define ANIM_DOWNLEFT  5
#define ANIM_LEFT      6
```

Poll

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
#define ANIM_UPLEFT    7

// Global Variables
u16 dir_plane = ANIM_RIGHT; // Direction the plane is facing
u8 plane;                   // Sprite object number of the plane
u8 plane_x = 110;           // X position of the plane
u8 plane_y =  50;           // Y position of the plane

// Function Prototypes
void vbl_func();          // VBL function
void query_buttons();     // Query for user input
void update_plane_gfx(); // Apply plane's new graphic
void update_plane_pos(); // Apply plane's new position

// Function: main()
int main()
{
    // Variables
    map_fragment_info_ptr bg_water; // Pointer to our background

    // Initialize HAMlib
    ham_Init();

    // Initialize the text display system
    ham_InitText(2);

    // Setup the background mode
    ham_SetBgMode(1);

    // Initialize the palettes
    ham_LoadBGPal((void*)background_Palette,256);
    ham_LoadObjPal((void*)object_Palette,256);

    // Setup the tileset for our image
    ham_bg[0].ti = ham_InitTileSet((void*)water_Tiles,
        SIZEOF_16BIT(water_Tiles),1,1);

    // Setup the map for our image
    ham_bg[0].mi = ham_InitMapEmptySet(3,0);
    bg_water = ham_InitMapFragment((void*)water_Map,
        30,20,0,0,30,20,0);
    ham_InsertMapFragment(bg_water,0,0,0);

    // Display the background
    ham_InitBg(0,1,0,0);

    // Setup the plane
    plane = ham_CreateObj((void*)red_plane_anim_Bitmap,
        0,3,OBJ_MODE_NORMAL,1,0,0,0,0,0,0,plane_x,plane_y);

    // Start the VBL interrupt handler
    ham_StartIntHandler(INT_TYPE_VBL,(void*)&vbl_func);

    // Infinite loop to keep the program running
```

```
    while(1) {}

    return 0;
} // End of main()

//**********************
// Function: vbl_func()
// Purpose: VBL function
//**********************
void vbl_func()
{
    // Copy plane sprite to hardware
    ham_CopyObjToOAM();

    // Process the following every VBL
    query_buttons();    // Query for user input
    update_plane_gfx(); // Apply plane's new graphic
    update_plane_pos(); // Apply plane's new position

    return;
} // End of vblFunc()

//*****************************
// Function: query_buttons()
// Purpose: Query for user input
//*****************************
void query_buttons()
{
    // UP only
    if(F_CTRLINPUT_UP_PRESSED
        && !F_CTRLINPUT_RIGHT_PRESSED
        && !F_CTRLINPUT_LEFT_PRESSED)
    {
        if (plane_y > 0) plane_y--;
        dir_plane = ANIM_UP;
        return;
    }

    // UP + RIGHT
    if(F_CTRLINPUT_UP_PRESSED
        && F_CTRLINPUT_RIGHT_PRESSED)
    {
        if (plane_y > 0) plane_y--;
        if (plane_x < 176) plane_x++;
        dir_plane = ANIM_UPRIGHT;
        return;
    }

    // RIGHT only
    if(F_CTRLINPUT_RIGHT_PRESSED
        && !F_CTRLINPUT_UP_PRESSED
        && !F_CTRLINPUT_DOWN_PRESSED)
    {
        if (plane_x < 176) plane_x++;
```

```
        dir_plane = ANIM_RIGHT;
        return;
    }

    // DOWN + RIGHT
    if(F_CTRLINPUT_DOWN_PRESSED
        && F_CTRLINPUT_RIGHT_PRESSED)
    {
        if (plane_y < 96) plane_y++;
        if (plane_x < 176) plane_x++;
        dir_plane = ANIM_DOWNRIGHT;
        return;
    }

    // DOWN only
    if(F_CTRLINPUT_DOWN_PRESSED
        && !F_CTRLINPUT_RIGHT_PRESSED
        && !F_CTRLINPUT_LEFT_PRESSED)
    {
        if (plane_y < 96) plane_y++;
        dir_plane = ANIM_DOWN;
        return;
    }

    // DOWN + LEFT
    if(F_CTRLINPUT_DOWN_PRESSED
        && F_CTRLINPUT_LEFT_PRESSED)
    {
        if (plane_y < 96) plane_y++;
        if (plane_x > 0) plane_x--;
        dir_plane = ANIM_DOWNLEFT;
        return;
    }

    // LEFT only
    if(F_CTRLINPUT_LEFT_PRESSED
        && !F_CTRLINPUT_UP_PRESSED
        && !F_CTRLINPUT_DOWN_PRESSED)
    {
        if (plane_x > 0) plane_x--;
        dir_plane = ANIM_LEFT;
        return;
    }

    // UP + LEFT
    if(F_CTRLINPUT_UP_PRESSED
        && F_CTRLINPUT_LEFT_PRESSED)
    {
        if (plane_y > 0) plane_y--;
        if (plane_x > 0) plane_x--;
        dir_plane = ANIM_UPLEFT;
        return;
    }
```

```
    return;
} // End of query_buttons()

//**********************************
// Function: update_plane_gfx()
// Purpose: Apply plane's new graphic
//**********************************
void update_plane_gfx()
{
    ham_UpdateObjGfx(plane,
        (void*)&red_plane_anim_Bitmap[4096*dir_plane]);

    return;
} // End of update_plane_gfx()

//**********************************
// Function: update_plane_pos()
// Purpose: Apply plane's new position
//**********************************
void update_plane_pos()
{
    ham_SetObjX(plane,plane_x);
    ham_SetObjY(plane,plane_y);

    return;
} // End of update_plane_pos()
```

Code Explanation

```
// Defines
#define ANIM_UP         0
#define ANIM_UPRIGHT    1
#define ANIM_RIGHT      2
#define ANIM_DOWNRIGHT  3
#define ANIM_DOWN       4
#define ANIM_DOWNLEFT   5
#define ANIM_LEFT       6
#define ANIM_UPLEFT     7
```

(I'll assume you know what defines are.) What I am doing is setting up a way to keep track of direction and we'll use this when loading the image that corresponds to the direction.

// Global Variables
These should all make sense to you.

// Function Prototypes
These are very similar to those from Day 7.

int main()
You should notice that almost everything is the same as Day 7 as well except for the following code.

// Setup the plane
plane = ham_CreateObj(...)
This call is almost identical to the one used in Day 5 except that the shape and size are 0,3 this time. That corresponds to an 8x8 8 pixel square, which just means a 64x64 pixel square. As I said earlier, that is what the plane sprite is.

void vbl_func()
Again, this is the similar to Day 7 except that I added a function to update the plane's sprite depending on the direction.

void query_buttons()
This function should be pretty easy to figure out. I know that this function could be implemented a little differently, but I am going for ease of understanding right now and not necessary all-out speed. You might wonder where I got the 176 & 96. Easy!
240 (screen width) - 64 (sprite width) = 176
160 (screen height) - 64 (sprite height) = 96

update_plane_gfx()
ham_UpdateObjGfx(...);
This call is used to update a sprite. Basically our plane sprite index will be updated with the part of the image that corresponds to the direction in which the plane is flying. For example, I defined *dir_plane = ANIM_RIGHT* at the beginning of the code because I want the plane to start out flying to the right. (Notice *ANIM_RIGHT = 2*). Now, 'where does the 4096 come from?' you ask. Remember that the sprite is 64x64 pixel square and 64 * 64 = 4096. That means that every 4096 spots in the array corresponds to one direction for the plane. 'Huh?' you ask. Well, take a look at the following:
red_plane_anim_Bitmap[0] corresponds to the plane flying straight up.
red_plane_anim_Bitmap[4096] corresponds to the plane flying diagonally up and to the right.
red_plane_anim_Bitmap[8192] corresponds to the plane flying to the right.
... and so on ...
So, *dir_plane = 2* and 4096 * 2 = 8192. Therefore we update *plane* with the data starting at *red_plane_anim_Bitmap[8192]*.

One final note on this function call. You'll notice the *(void*)* before *&red_plane_anim_Bitmap[...]*. HAM will handle the function call without *(void*)*, and your code will compile, it just makes the error messages disappear.

void update_plane_pos()
Another function that is almost identical to Day 7.

Well, that's it for Day 8, isn't that great, can't wait ... for Day 9.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.80 And Higher
Download All Files In One Zip: Day8_Sprites2_Animation.zip

View Demo Now

Discuss Day 8

<< HOME >>