

Introduction to the Propeller Animation & Video Game Programming Chapter 2

By Jeff Ledger ‘Oldbitcollector’
March 2010 (pre-release 1.0)

In the last chapter we introduced Propeller basics, screen display, and character editing. This time we’ll turn things up a bit and get into animation and simple game controls.

Requirements:

A Propeller board with NES controls or keyboard interface & video output.
A connected TV or composite monitor.
OREDemo2.zip from <http://www.orrtech.us/oredemo2.zip>

Download and unzip the OREDemo2.zip file into its own folder. It contains everything we’ll need for this chapter, including the character editing tools from the last chapter.

Open the program “CHARACTER DEMO” with the Propeller Tool.
Before we actually run this program with the F10 key, let’s examine the I/O setting to make sure it’s compatible with your Propeller setup.

```
CON
* ====={ Speed & I/O Settings }=====

  _clkmode  = xtall + pll16x
  _xinfreq  = 5_000_000

  TV_BASE_PIN = 12
  KEYBOARD    = 26

* Gamepad pin configuration

  NES_CLK = 3
  NES_LCH = 4
  NES_DATAOUT0 = 5
  NES_DATAOUT1 = 6

* =====
```

This configuration will work perfectly with Propeller Demoboard & compatible configurations.

If you don’t have a NES controller interface, simply plug in a keyboard and use the arrow keys to control this demo.

By changing the value numbers in blue we can adjust the configuration for any board which has a Propeller.

Here are the settings for the Hydra and El Jugador boards:

Hydra Changes: (NES settings left the same.)

```
_CLKMODE = xtal1 + pll8x  
_XINFREQ = 10_000_000 + 0000
```

```
TV_BASE_PIN = 24  
KEYBOARD    = 13
```

' Gamepad pin configuration

```
NES_CLK = 3  
NES_LCH = 4  
NES_DATAOUT0 = 5  
NES_DATAOUT1 = 6
```

Propeller Platform w/ El Jugador: (KEYBOARD unused, set as 0)

```
_CLKMODE = xtal1 + pll16x  
_XINFREQ = 5_000_000
```

```
TV_BASE_PIN = 12  
KEYBOARD    = 0
```

' Gamepad pin configuration

```
NES_CLK = 23  
NES_LCH = 24  
NES_DATAOUT0 = 25  
NES_DATAOUT1 = 26
```

Popular Propeller Boards:

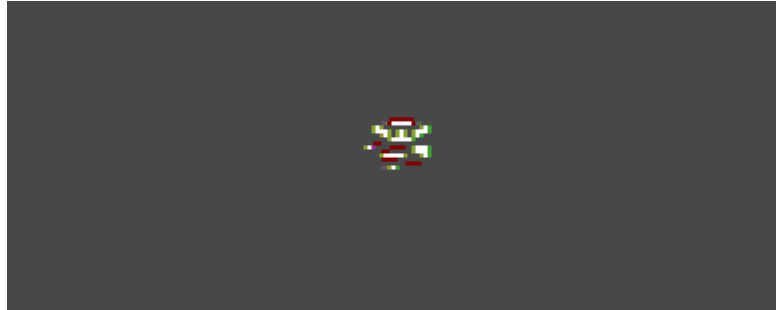
The [Hydra Game Development Kit](#) are in stock at Parallax.com.

The [El Jugador](#) & [Propeller Platform](#) are in stock at Gadgetgangster.com

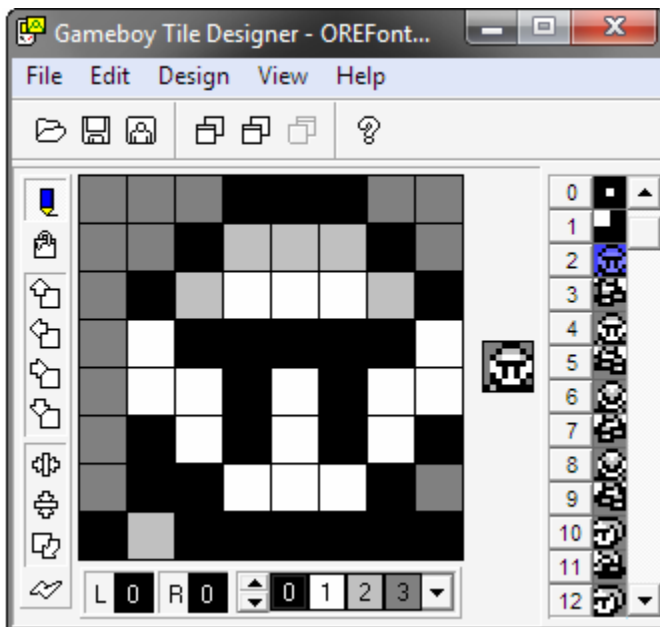
SpinStudio & Game Stacker are in stock at [uController.com](#)

Once you have your settings correct, upload the program using the **F10** key.

You should see a screen like this one:



Pressing either the arrow keys on the keyboard or NES controller will move our character up and down the screen. If you look carefully, you'll see two 8x16 characters used for forward animation, and two others used for backward animation. Page flipping multiple characters gives the illusion of animation.



Opening the **OREFont** in Gameboy Tile Designer we can see the four 8x16 combinations used in our animation.

2-3 & 4-5 are used when going forward.

6-7 & 8-9 are used when going backward.

Every time we press on the controls the screen is cleared and alternating versions of the forward or backward characters are displayed, at the new screen position.

Let's break the program down section by section.

```
' NES bit encodings for NES gamepad 0
NES0_RIGHT    = %00000000_00000001
NES0_LEFT     = %00000000_00000010
NES0_DOWN     = %00000000_00000100
NES0_UP       = %00000000_00001000
NES0_START    = %00000000_00010000
NES0_SELECT   = %00000000_00100000
NES0_B        = %00000000_01000000
NES0_A        = %00000000_10000000

' Define colors
COLOR_BLACK   = $02
COLOR_BROWN  = $5d
COLOR_YELLOW  = $6e
COLOR_WHITE   = $07
COLOR_GREEN   = $ab
COLOR_LGREEN  = $9e
COLOR_RED     = $4b
COLOR_PURPLE  = $2b
COLOR_CYAN    = $58
```

This section sets up the settings for the NES gamepad.

No changes are typically required here.

It's easier to work with color names instead of the hexadecimal numbers so we assign some easy names in this section.

About color:

The ORE driver is capable displaying 126 colors. Most of the time we use only a handful of colors, but you can add more using the following chart.

\$02, \$03, \$04, \$05, \$06, \$07	
\$19, \$1a, \$1b, \$1c, \$1d, \$1e, \$98, \$af	Magenta
\$29, \$2a, \$2b, \$2c, \$2d, \$2e, \$a8, \$bf	
\$39, \$3a, \$3b, \$3c, \$3d, \$3e, \$b8, \$cf	
\$49, \$4a, \$4b, \$4c, \$4d, \$4e, \$c8, \$df	Red
\$59, \$5a, \$5b, \$5c, \$5d, \$5e, \$d8, \$ef	
\$69, \$6a, \$6b, \$6c, \$6d, \$6e, \$e8, \$ff	Orange
\$79, \$7a, \$7b, \$7c, \$7d, \$7e, \$f8, \$0f	
\$89, \$8a, \$8b, \$8c, \$8d, \$8e, \$08, \$1f	
\$99, \$9a, \$9b, \$9c, \$9d, \$9e, \$18, \$2f	Green
\$a9, \$aa, \$ab, \$ac, \$ad, \$ae, \$28, \$3f	
\$b9, \$ba, \$bb, \$bc, \$bd, \$be, \$38, \$4f	
\$c9, \$ca, \$cb, \$cc, \$cd, \$ce, \$48, \$5f	
\$d9, \$da, \$db, \$dc, \$dd, \$de, \$58, \$6f	Blue
\$e9, \$ea, \$eb, \$ec, \$ed, \$ee, \$68, \$7f	
\$f9, \$fa, \$fb, \$fc, \$fd, \$fe, \$78, \$8f	

```
'direction constants
DIR_SOUTH = 2
DIR_NORTH = 6
DIR_WEST  = 10
DIR_EAST  = 14
```

These four lines of code assign the initial character to display in each direction.

The number correspond with the characters from GBTD,

All of the yellow section at the top of the program is referred to as the **CON** or “Constants” section. It is used to define all of the settings which will not change when the program is run. This is why I/O settings and speed settings are always defined in this section.

Now let’s take a look at the **VAR** section:

```
VAR
word Nes_Pad

byte player_dir      'the player direction
byte player_step     'the step the player is on
byte player_x        'x position
byte player_y        'y position

byte player_color_1  'player color 1
byte player_color_2  'player color 2
byte player_color_3  'player color 3
```

The **VAR** section defines variable placeholders we’ll use in the program code. You can think of **byte** definitions in the **VAR** section as empty boxes in which we will place information {variables} later.

Reminder: Any text after the ‘ mark is a program comment.

The **OBJ** or “Object” section calls on the ORE driver and Controller driver. There is no need to alter these lines. Just make a mental note of the names: **text & controller**.

```
OBJ
text      : "ORE_TV_TEXT_010" 'the TV display driver
controller : "Game_controller" 'the Game Controller w/ Keyboard interface
```

Up to this point, we've defined items which won't change in the **CON** section, set up our empty variable storage boxes in **VAR**, and **OBJECTS** which will do the behind the scenes work.

Now it's time to see the program itself.

```
PUB start

    'start the tv text driver
    text.start(TV_BASE_PIN)
    controller.start(NES_CLK,NES_LCH,NES_DATAOUT0,NES_DATAOUT1,KEYBOARD)

    'set up the initial character facing/location/colors
    player_dir      := DIR_SOUTH
    player_step     := 0
    player_x       := 9
    player_y       := 8
    player_color_1 := COLOR_WHITE      'Player skin color
    player_color_2 := COLOR_RED        'Player clothing color
    player_color_3 := COLOR_BLACK      'Player background color
```

The **PUB start** is the core of our program.

The first two lines, (text.start and controller.start) launch the two **OBJECTS** called with our **CON** definitions. There is no need to changes these lines.

Remember those empty byte boxes we defined in **VAR**?

Now we'll drop some information in those boxes to work with in the game.

```
    'set up the initial character facing/location/colors
    player_dir      := DIR_SOUTH
    player_step     := 0
    player_x       := 9
    player_y       := 8
    player_color_1 := COLOR_WHITE      'Player skin color
    player_color_2 := COLOR_RED        'Player clothing color
    player_color_3 := COLOR_BLACK      'Player background color
```

We can refill these variable boxes with new information anytime over the course of our program. Setting them up at the beginning of **PUB Start** gives us our starting points.

You should recognize some of our definitions from **CON**.

Try changing the **colors** or **player_y** definitions and see what happens.

Continuing further into PUB start, we get into the core loop of our program.

<pre>'set up the game loop repeat 'Clear the screen text.cls 'Check for gamepad events CheckGamePad 'draw the player DrawPlayer 'build in a delay for animation waitcnt(5_000_000 + cnt) 'update the screen with changes text.UpdateScreen</pre>	<p>All commands indented under the repeat command will be looped.</p> <p>text.cls clears the screen</p> <p>CheckGamePad to see if a key has been pressed</p> <p>Redraw the character</p> <p>Slow the loop down with a small delay.</p> <p>Update the screen with the changes.</p>
--	---

Now let's take a look at the additional code we are using in our repeat loop. **CheckGamePad** checks for input from the NES controllers & keyboard.

```
PRI CheckGamePad

Nes_Pad := controller.NES_Read_Gamepad

'gamepad has been moved down (south)
if (Nes_Pad & NES0_DOWN)
  player_dir := DIR_SOUTH
  player_y++

  'bounds check: if the player_y is > 16,
  'they would be 'out of bounds'
  if (player_y > 16)
    player_y := 16

  PlayerAnimate
```

PRI CheckGamePad defines this as an independent section.

The next line reads the waiting keypress or gamepad button.

```
Nes_Pad := controller.NES_Read_Gamepad
```

The `if (Nes_pad & NES0_DOWN)` starts the list commands to be done when DOWN is pressed. Notice how all commands are indented below it? The indented commands are activated each time DOWN is detected.

```
if (Nes_Pad & NES0_DOWN)
    player_dir := DIR_SOUTH

    player_y++

    'bounds check: if the player_y is > 16,
    'they would be 'out of bounds'
    if (player_y > 16)
        player_y := 16

    PlayerAnimate
```

Remember `player_dir := DIR_SOUTH` is the starting character to display. `player_y++` updates our screen position by adding 1.

The “bounds check” code maintains that our player position will never exceed 16. If `player_y` ever exceeds 16, it is set to 16. This keeps our player from going out of bounds.

PlayerAnimate determines which of the two characters to display when walking. Remember that by alternating between two 8x16 characters we create character animation.

I've only posted half of the **CheckGamePad** code. Take a look at the next section and see if you can determine how it works. It's literally the opposite of the code we just examined.

We'll save **PlayerAnimate & DrawPlayer** for the next chapter. In the meantime, spend some time with the code.

Thing to try:

Expand the movement limitations in `CheckGamePad`.

Change the character colors.

Change the character speed by increasing or decreasing delay,

Edit the character itself.