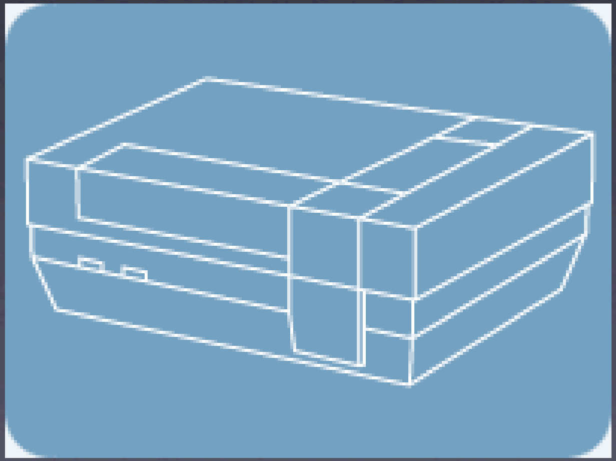
98-026
Nintendo

Bob Rost
January 14, 2004



Today

- Project Status
- Announcements
- Backgrounds
- PPU control registers
- Memory Mappers, Larger ROMs
- General Game Programming Tricks

Project Status

- Have you started?
- What are you doing?
- Demos!

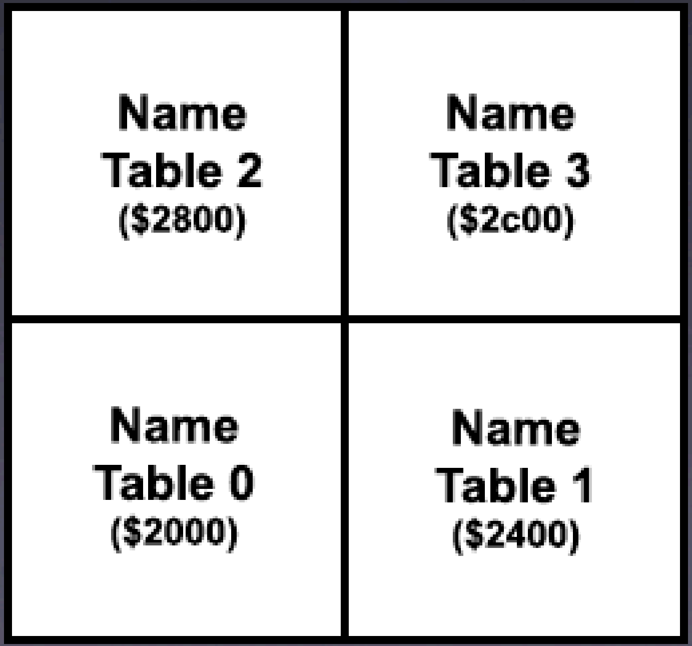
Announcement

- “Robort Bost” is going to the Game Developers Conference, March 23-27
- I’m trying to go in his place
- Cancelling class March 24th would be lame if some of you would like to present anything relevant
- Let me know if you would like to present something that day, and how much time you would want

Do they mean *this* Robort?

Backgrounds

- 4 Screen Buffers (Name Tables) (32x30 tiles each)
- Horizontal vs Vertical mirroring

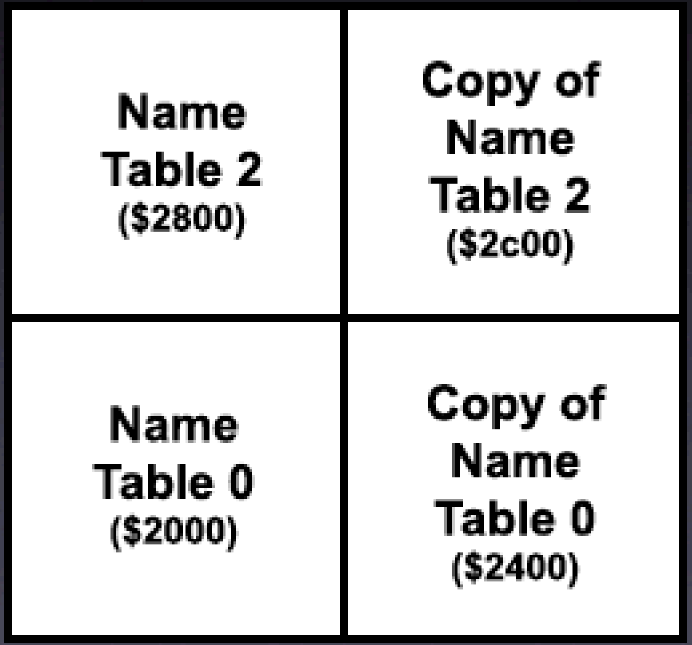


Name Tables

- Similar to sprites, but for the background
- 32x30 tiles per name table
- Each byte in the table is a reference to a tile in the pattern table
- Start writing PPU memory to each name table starting at the base address.
- Usually row order, but register $2000 will let you write in column order

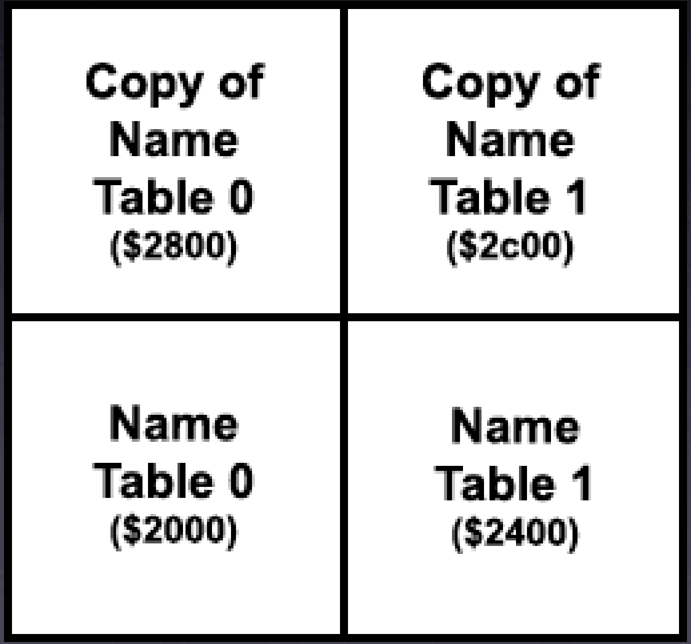
Horizontal Mirroring

- The NES only has enough VRAM for 2 Name tables
- Horizontal mirroring wires horizontally adjacent name tables together. *(Changing one affects the other)*
- Good for vertical scroller games like Spy Hunter



Vertical Mirroring

- Wires vertically adjacent name tables together
- Good for side scroller games, like Super Mario Bros.



Attribute Table

- 32 x 30 tiles is only 960 bytes
- 64 more and we’ve got a full kilobyte
- Let’s invent a difficult way to set colors in the background!

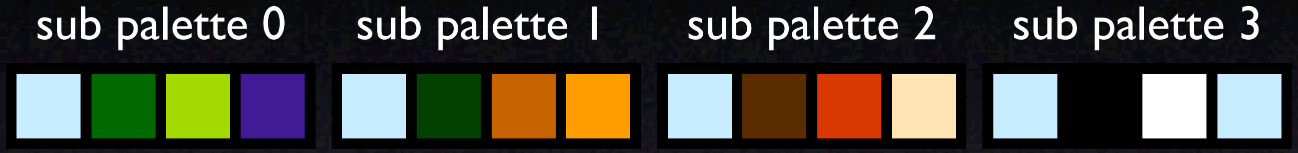
Attribute Table

- The attribute table determine the sub palettes of name table tiles
- Bytes are arranged row-order
- Each byte affects a 4x4 tile area of the background

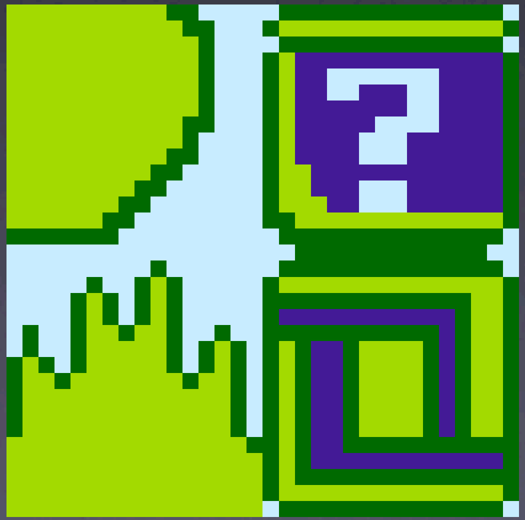
Attribute Byte

- Affects a 4x4 tile area of the name table
- 8 bits per byte, 2 bits to set a sub palette. A byte can set 4 sub palettes.
- Every 2 bits affects 2x2 tiles

Attribute Byte

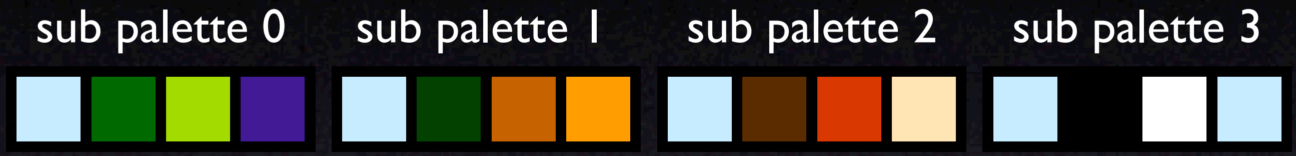


Value of attribute byte: 0
All tiles use sub palette 0



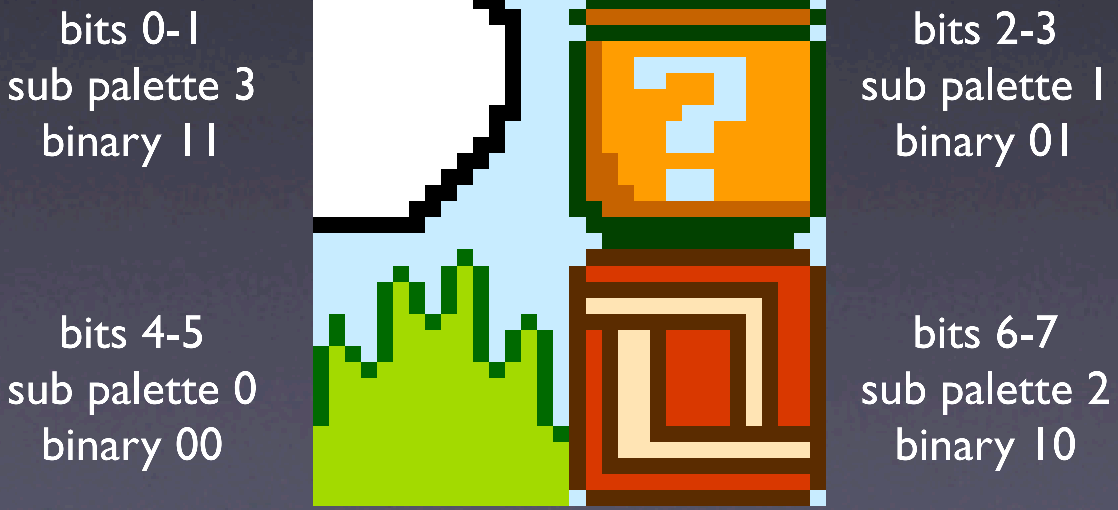
4x4 tiles (32x32 pixels)

Attribute Byte



Value of attribute byte: 135

Binary value: 10 00 01 11



Reference Slide

Name Table and Attribute Table memory locations (PPU memory)

Name Table 0	$2000
Attribute Table 0	$23c0
Name Table 1	$2400
Attribute Table 1	$27c0
Name Table 2	$2800
Attribute Table 2	$2bc0
Name Table 3	$2c00
Attribute Table 3	$2fc0

PPU Control Register $2000

- bits 0-1: Current base name table (for scrolling)
- bit 2: PPU Address Increment
  (0 = increment by 1, 1 = increment by 32)
- bit 3: Sprite pattern table
- bit 4: Background pattern table
- bit 5: Sprite size (0 = 8x8, 1 = 8x16)
- bit 6: unused
- bit 7: Execute NMI on Vblank (0=false, 1=true)

PPU Control Register
$2001

- bit 0: Display type (0 = color, 1 = monochrome)
- bit 1: Background clipping (0 = don’t show left 8 pixels)
- bit 2: Sprite clipping (0 = invisible in left 8 pixel column)
- bit 3: Background visiblity (0 = don’t show, 1 = show)
- bit 4: Sprite visibility (0 = don’t show, 1 = show)
- bits 5-7: Color emphasis bits

Memory Mappers

- iNES Header
- Bank Switching
- MMC3

iNES Header

- Information about the ROM
  Number of PRG banks (16k each)
  Number of CHR banks (8k, 2 tables each)
  Mirroring Type (0 = horizontal, 1 = vertical)
  Memory Mapper (0 = none, 4 = MMC3)
- Check out *header.bas* in the demo ROMs

Bank Switching

- The NES can only address 64k of memory
- Only 32k is for ROM (code and data)
- Large games need to be able to swap areas of memory in and out
- Usually have a 16k bank for code and swap 16k of level data
- Memory mappers let us swap the memory

Bank Boundaries

- In ROM source, you may use assembly to declare bank boundaries
- Banks usually have a base address of $8000 or $C000

```
asm
        .org $8000 //base address
        .bank 0 //0 = first bank
endasm
```

MMC3

- iNES Mapper #4
- Extra memory capability for battery-backed save RAM
- Allows swapping 8k blocks of PRG memory
- Allows swapping 64- or 128-tile blocks of CHR pattern table
- Allows name table mirror type switching
- Common, versatile, and easy to use
- SMB2, SMB3, SOF, Megaman 3-6, many others

MMC3

- Splits PRG region ($8000-$ffff) into 4 banks of 8k each
- You can swap the lower banks or the middle banks with other ROM banks. Upper bank is hardwired to the last bank in the ROM.
- I suggest you locate your game code in the upper banks and swap the lower banks for level data

MMC3 Usage

- Support code on Resources page
- Example to swap lower 8k of PRG memory

```
gosub mmc3_use_lower_banks //use lower banks
set mmc3_command 6 //swap to first 8k bank
set mmc3_pagenum 0 //copy from first 8k of ROM
gosub mmc3_execute_command //swap
```

MMC3 Usage

- Example to swap first 128 tiles of first pattern table
- Read the source file to see available commands and functions

```
set mmc3_command 0 //swap 2k page to PPU $0000
set mmc3_pagenum 4 //let's copy from here
gosub mmc3_execute_command //go!
```

Game Programming Tricks

“The ghettoer the bettoer”

Game Programming Tricks

- Animation
- Table Lookups
- Multiplication
- Gravity Physics
- Collision Detection
- Random Numbers
- Multi Screen Tips

I really doubt we have time for all of these today

Animation

Super Mario 3
Overhead Map

How many sprites?

Background Animation

- The MMC3 can swap 64 or 128 tiles at a time in the pattern table
- ...
- Profit!

Foreground Animation

- The MMC3 lets us swap portions of the pattern table. Sounds familiar...
- You can also use the same trick to give your character new clothes for free (Raccoon Mario, Tanuki Mario, Hammer Mario, SOF with a space helmet)

Table Lookups

- Sometimes you need complex calculations
- These are hard and slow on the NES
- In many cases, you can pre-compute values and store them in a static array, then just look in the array when you need the value
- Useful for physics, complex movements, exponential decay...

Multiplication

- The 6502 does not support multiplication or division by anything other than 2
- Try the "Russian Peasant" method
- Google will help you out

Gravity

- Medieval physics
- Newtonian physics

Medieval Gravity

- Impetus, constant speed ascent and fall
- Move up a pixel per frame until you run out of impetus. Move down a pixel per frame until you hit the ground
- Fine for games with falling but not jumping (Gyromite, Donkey Kong Jr, Dig Dug)
- Sometimes tolerable for jumping

Newtonian Gravity

- Newton told us that things going through the air move in a parabola
- We can't just "move some number of pixels each frame"
- It's hard to calculate a parabola on the 6502

Newtonian Gravity

- Use a lookup table
- Store delta movement for each moment in time during a jump
- Two ways to do this...

Newtonian Gravity

Method 1

- Array values are between 0 and max speed per frame (4?)
- Keep track of direction and location in array, move array index each frame
- For a jump, immediately move near the end of the array
- Go back to 0 when we hit the ground

```
gravity:
  data 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1
  data 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1
  data 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
  data 3, 3, 3, 3, 3
```

Newtonian Gravity
Method 2

- Array values are between 0 and max speed, and between 255 and 255-max speed
- Adding large numbers is the same as subtracting small ones
- Add array value and move array index each frame
- Go back to array center when we hit the ground

```
gravity:
  data 253, 253, 254, 253, 254, 254, 254, 254, 254, 254
  data 254, 254, 255, 254, 255, 254, 255, 255, 255, 255
  data 255, 0, 255, 255, 255, 255, 0, 255, 0, 255, 0, 0, 0
  //array center. no movement
  data 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1
  data 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 3, 3
```

Collision Detection

- Want to know when objects collide with the world, or with each other
- It’s difficult and impractical to do it per pixel
- Use simple shapes, like rectangles
- Modern games still use similar methods

Collision With Ground

Simple Method

- Keep track of how high the ground is
- Keep track of how high your object is
- If object_y >= ground_y, it hits the ground
- Move your object to just above ground height and reset the gravity array index for that object
- Works well for very simple worlds, or falling down pits of death

Collision With the World

- Make a collision map, a large array corresponding to the background screen that tells whether each block is impassable
- Update the collision map as the game scrolls
- Test an object’s bounding rectangle against the collision map

Collision Between Objects

- Give each object a bounding rectangle. Remember that the object height may be different than the sprite size. (SOF is 22 pixels tall, but his sprites are 32 pixels tall)
- Usually you want to keep track of the direction each was moving, to know if the goomba should die or Mario should get hurt

Random Numbers

- Hard to make on your own
- Create from player input?
- Table lookup?
- Pseudo-random function?

Random Numbers

- I posted a pseudo-random function on the webpage. *(Did anyone test it?)*
- Setting the random seed, looping on the start screen

```
set random_seed 1 //anything non-zero
gosub random_number
//register A now holds a random number
set my_var a
```

Multi Screen Tips

- Updating the name table during gameplay is hard.
- It’s easier to restrict your game to 2 screens at a time. (go through doors to other sections?)
- Infinite scrolling backgrounds are good in space shooters

GAME OVER