The Making of Super Mario Clouds – BY CORY ARCANGEL (BEIGE)

SECTION 1:
SOURCE CODE AND COMMENTARY

In 2002, I posted a tutorial on the Internet about a project I made called "Super Mario Clouds". In this work, I took an old Super Mario Brothers Nintendo video game and erased everything but the clouds. The tutorial explained my motives for making this work, and included the source code behind the work as well as technical information on how the work was constructed. Over the years, I have gotten many emails about this web page with technical questions (people of course were free to use these directions to make their own Super Mario Clouds), and I have realized there were many parts I could have written better (especially the technical section). So I am pleased to say that below I have rewritten this page, and have attempted to address the questions I have received from people over the years. Also, as a side note, I actually updated the code to this project in 2003 (my laptop was stolen... it was a big mess), therefore I have updated the code below as well.

Below I will go through the source code line by line. The idea here is to simply write a program that will take the clouds from Super Mario Brothers and scroll them across the screen. Here comes the first line of code:

```
;**************************
;
; Super Mario Clouds
;
; BEIGE 2002/03
; Cory Arcangel
;
; www.post-data.org/beige/
; www.beigerecords.com/cory/
; del.icio.us/cory_arcangel/
;
; "you mess with the best,
; you die like the rest"
; - Anon
;
; "punks jump up to get beat
; down"
; - Brand Nubian
;
 **************************
```

When computer code is made public it is common for programmers to give contact information for a variety of reasons. Here I have also included a few small phrases aimed at media artists who think they can step to my style. This is a trait I inherited from the early commodore 64 cracking scene where gangs of hackers would compete to crack video games (crack means to remove copyright protection), and they would also make very elaborate video introductions in order to show off. These often contained jibes at other competing groups.

Before I get started with the code I would like to take this opportunity to state that I am not really a programmer. The first time I took a class in "computer science" was at a summer school when I was 8 or 9 years old, and I remember crying and switching to the "storytelling" class. Years later in college I still didn't like computer science and got below 50% in most of my exams. I have since grown used to programming only because it is the mechanism that seems to make most of the world move. Believe me, if I could order pizzas (dominos has a great online delivery mechanism) by painting, I definitely would paint. For those who are interested, I have actually hacked the dominos pizza ordering system in a project called "pizza party" with help from my super-genius programming friend Michael Frumin. Google "pizza party"... last time I checked it was the second result. So let's get back on track, the first line in this program that actually does anything is:

asm

This program is written in a programming language called assembly. Assembly language is the lowest level someone can program. It is one step away from the ones and zeros, and in some cases involves actual ones and zeros. I tend to prefer assembly because it gives me control over the machine and assures me that aesthetic choices are based on the hardware of the machine and not, say, some dupe at Macromedia. The above line of code simply says we will be writing this program in assembly. The particular brand of assembly we will be using is 6502 assembly. That is because the Nintendo runs on the 6502 microprocessor. The 6502 is the chip that made the Apple II possible and thus revolutionized home computing. The Nintendo runs on a modified version of this chip. It should not be surprising that the Nintendo and Apple run on the same processor, because video game systems are really just home computers without disk drives.

```
.inesprg 2
.ineschr 1
.inesmir 1
.inesmap 0

.org $8000

clouds_start:
.dw clouds_start_addr
clouds_start_addr:
.incbin "clouds.hex"
```

I like the idea of making things out of trash (one can easily find an NES in a dumpster these days) and I like the idea of actually having to break into something that I find in the trash even better. The only way to make work for the NES is to hack and solder a cartridge. To do this, I will clip off the program chip from an actual Mario cartridge, burn my new information, the output of this code compiled to a chip, and solder it in the place of the old one. Please see SECTION 2 of this tutorial for information on how to do this. The above lines basically set up our compiler and tell it that we would like this program to be for a chip which is 32k (the same as an old Super Mario Brothers chip), and also that we will later be working with a file called "clouds.hex", which is a data file of the clouds.

The next part of code tells the cartridge what to do when we press the reset button on the NES. This code is modified from a Canadian NES genius named Chris Covell, who apparently got it from Duck Hunt. Awesome. I learned to program in assembly language looking at examples of code posted by Chris Covell, and as with a lot of this 8-bit work information comes mostly from the hobby scene. In my opinion these are the true heroes of contemporary computer art. Out of the hobby scene have come portable play stations, Dreamcasts that boot LINUX, and even hard drives that play music by spinning at different speeds.

```
start:

     sei
     cld

.vblank_clear1
     lda $2002
     bpl .vblank_clear1

.vblank_clear2
     lda $2002
     bpl .vblank_clear2

.vblank_clear3
     lda $2002
     bpl .vblank_clear3

     sei
     cld

.vblank_clear4
     lda $2002
     bpl .vblank_clear4

.vblank_clear5
     lda $2002
     bpl .vblank_clear5

.vblank_clear6
     lda $2002
     bpl .vblank_clear6

     lda 0
     ldx 0
.clear_out_ram
     sta $000,x
     sta $100,x
     sta $200,x
     sta $300,x
     sta $400,x
     sta $500,x
     sta $600,x
     sta $700,x
     inx
     bne .clear_out_ram

     lda 0
     ldx 0
.clear_out_sprites
     sta $2000,x
     sta $2100,x
     sta $2200,x
     sta $2300,x
     sta $2400,x
     sta $2500,x
     sta $2600,x
     sta $2700,x
     sta $2800,x
     sta $2900,x
     sta $2a00,x
     sta $2b00,x
     sta $2c00,x
     sta $2d00,x
     sta $2e00,x
     sta $2f00,x
     inx
     bne .clear_out_sprites

     ldx #$FF
     txs

     jsr vwait
     jsr vwait
```

Ever wonder why Mario and Zelda were little squares? The Nintendo can only display graphics in 8 pixel by 8 pixel squares, and can only hold 8k of graphics in total, therefore Mario and Zelda were simply adhering to the hardware limitations of the Nintendo System. These two hardware limitations defined the aesthetic of most early 80s video games on the Nintendo, and making "art" for this system is a study of these limitations. Below I load up the color that this cartridge will use into the palette RAM. The Nintendo can only display 4 colors in any 8 by 8 square, for this cartridge I will use black, blue, light blue and white, which translates to $0d, $11, $21, $30 in the Nintendo palette.


```
;+++++++++++++++++++++++++++
; load palette
 +++++++++++++++++++++++++++

     lda #$3F
     sta $2006
     lda #0
     sta $2006

     lda #$21 ;background
     (powder blue)
     sta $2007
     lda #$30 ;cloud inside
     (blue)
     sta $2007
     lda #$11 ;highlight
     (blue)
     sta $2007
     lda #$0d ;outline
     (black)
     sta $2007
```

A typical NES Cartridge has two chips. One is a graphics chip, and the other is a program chip. Basically the program chip tells the graphics chip where to put the graphics, and thus if you do this in an interesting manner, you have a video game. When making a "Super Mario Clouds" cartridge, I only modify the program chip, and I leave the graphic chip from the original game intact. Therefore, since I do not touch the graphics from the original cartridge, the clouds you see are the actual factory soldered clouds that come on the Mario cartridge. There is no generation loss, and no "copying", because I did not even have to make a copy. Wasss up.

The code below is where I load up the clouds. This referenced the file (clouds.hex) we pointed to earlier in the program.

```
;+++++++++++++++++++++++++++
; load name tables
 +++++++++++++++++++++++++++

endasm
array addr 2
set songloadloop 0
asm

     ldx #0
     ldy #0

     lda #2
     sta songloadloop
load_outcast:
     lda clouds_start,y
     sta addr,x
     iny
     inx
     dec songloadloop
     bne load_outcast
     lda #$20
     sta $2006
     lda 0
     sta $2006

nametables:

     ldx #$08
     ldy #$00

load_clouds:
     lda (addr),y
     sta $2007
     iny
     bne load_clouds
     txs
     ldx #$01
     inc addr,x
     tsx
     dex
     bne load_clouds

lda #$00
sta $2006
sta $2006 ;Reset PPU
sta $2005
sta $2005 ;Reset Scroll
```

For this cartridge I simply draw everything and then turn the screen on. The NES cannot just draw a picture to the screen like a modern computer, in fact to make something that looks similar on a modern computer would take about 3 minutes in Photoshop . It is too slow for that, so in order to change backgrounds one has to turn the screen off, draw a new picture, and turn it back on again. This is why the screen goes black for a split second between worlds on most early NES games. I turn the screen on by placing a sequence of ones and zeros into a specific Nintendo memory location. One highlight of working in assembly language is being able to actually use ones and zeros. So yeah, above I just drew everything and below in the section "init graphic settings", I turn on the screen!

```
;++++++++++++++++++++++++++++
; Init graphic settings
 ++++++++++++++++++++++++++++

lda #%10011000
sta $2000
lda #%00001010
sta $2001

;++++++++++++++++++++++++++++
; set variables for scroll
 ++++++++++++++++++++++++++++

endasm

      set DELAYSCROLL 1
      set NTShow 0
      set Scroll 0

asm
```

```
;++++++++++++++++++++++++++++
; loop forever...
 ++++++++++++++++++++++++++++

main:

jmp main

;++++++++++++++++++++++++++++
; vblankzzzzzz
 ++++++++++++++++++++++++++++

vwait:
      lda $2002
      bpl vwait
vwait2:
      lda $2002
      bmi vwait2
rts
```

That's it for the intro code. Now below I make it scroll.

A TV works by drawing a picture faster than your eye can see, every 1/60th of a second. In Europe it is 1/50th. Every time this scan line gets to the bottom of the screen it has to jump back to the top. This is called a vertical blanking interrupt. The NES can only draw graphics to the screen when this line is jumping from the bottom of the screen to the top. Below is the code which scrolls the clouds during this period when the electron beam is jumping. A highlight of working on early games systems is this intimate access to the display mechanism of the television.


```
;++++++++++++++++++++++++++++
; NMI Routine!!!!
; Very important!!!!
 ++++++++++++++++++++++++++++

nmi:

      dec DELAYSCROLL
      bne .end_no_scroll

      lda #$20
      sta DELAYSCROLL

      lda #$ff
      cmp $24 ;scroll
      beq .NT_adj
      jmp .end

.NT_adj:

      lda 0
      cmp NTShow
      beq .Show_Zero

      lda #%10011100
      sta $2000
      lda #%00001010
      sta $2001

      lda 0
      sta NTShow

      jmp .end

.Show_Zero:
      lda #%10011101
      sta $2000
      lda #%00001010
      sta $2001

      lda #$01
      sta NTShow

.end:

      inc $24 ;scroll
      lda $24 ;scroll
      sta $2005
      lda 0
      sta $2005

.end_no_scroll:

      rti

irq:

      rti
```

That's all the code. Now we just need to set the vector table appropriately. This tells the cartridge what to do when you put it in.

```
;++++++++++++++++++++++++++++
; Load Data Filez
 ++++++++++++++++++++++++++++

      .bank 3
      .org $fffa
      .dw nmi ;//NMI
      .dw start ;//Reset
      .dw irq ;//BRK
      .bank 4
      .org $0000
      ;//end of file

.incbin "mario.chr" ;
gotta be 8192 bytes long ;
include the graphics
for our emulator.

endasm
```

The end. The cartridge is complete. Not that bad. Actually programming for the NES is pretty simple. Now I would compile this program, and then burn it to a chip. Check SECTION 2 for this part!

SECTION 2:
HOW TO MAKE YOUR OWN
SUPER MARIO CLOUDS

This tutorial will teach you how to make your own Super Mario Clouds. The process of making a NES cartridge was orginally taught to me by BEIGE member Paul B. Davis. In this section I will pass this technique on to you. Basically in short, the process will entail us taking apart an old Super Mario Brothers video game cartridge, desoldering one of the chips in the cartridge, and then soldering a new one in its place.

a The first thing you will need to get is an original Super Mario Brothers cartridge. Not a "Duck Hunt+Mario Brothers" cartridge, but just a plain old Super Mario Brothers cartridge.

Next you should unscrew the plastic back on the cartridge, and inside you will see a circut board like the one you see below. There are two chips on this board. The CHR chip, and the PRG chip. We are interested in the PRG chip for this project. Also please make sure the cartridge says NES-NROM-01 (01-05 in also fine). This let's us know it is a 32k Nintendo circut board.

b Next take some wire clippers and clip the legs of the PRG chip. I like to use the red clippers from Radioshack.

c Once the legs are clipped you should be able to take the chip off like this.

d The end of the legs will still be soldered to the circut board though.

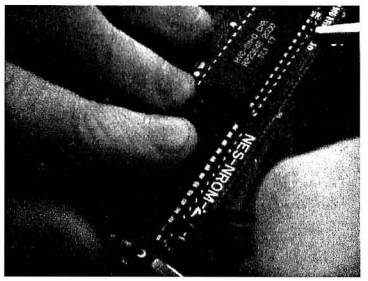
e Now, with a pair of wire holders, hold a leg that is still attached to the board. While you are holding it, touch it with a hot soldering iron. This will melt the wire that is keeping it attached to the board. You should feel the leg losen, and you will be able to pull it out of the circut board. Do this for each leg.

f Next, get some desoldering braid!

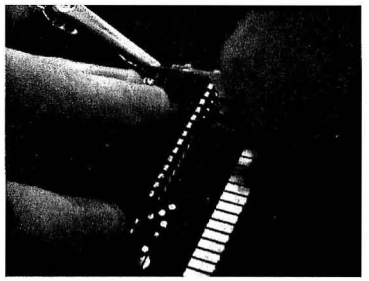
g Put it over the holes in the circut board. Then place the soldering iron on the braid. This will make the solder heat up and the braid will suck it up.

a

b

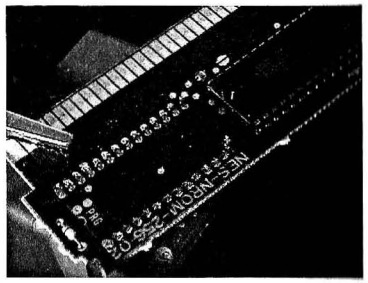


b

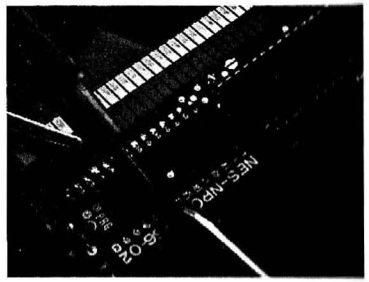


c

d

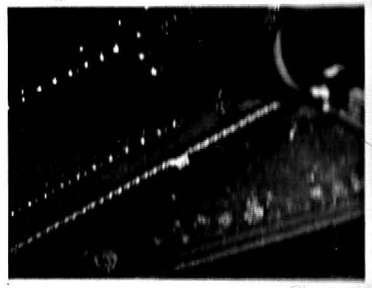


e



f

g



h When you are done, the holes in the circut board should look clean like this

g Now you will need to solder a socket into the holes where the program chip used to be. You will need to buy a 28-pin low profile sockets. You can get these form Jameco.com. This makes it so you can take the chip in and out of the socket without resoldering. This isn't so necessary, but I always make mistakes, so it is kind precaution in case I need to make a new chip.

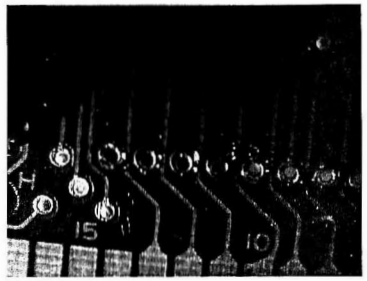
J To solder, just touch the pin and the solder at the same time, and you will see the soldering melt into the hole thus sealing the socket pin into the hole.

k Now you can place a chip into the hole. The way you make these chips is by getting an EEPROM burner. It is like a CD burner, except for computer chips as oppoed to Cd's. I would get one from Jameco. The kinda chip you will get is called a 27C256. This is a 32k EEPROM which is exactly the same one the Nintendo used for cartridges like Mario Brothers. I would also get those from Jameco.com. Remember to make sure your EEPROM burner can burn type 27C256 chips! So once you have this, (or maybe a friend has one?), you can download the compiled 32k file from www.belgarecords.com/corey/ and burn this data to the chip.

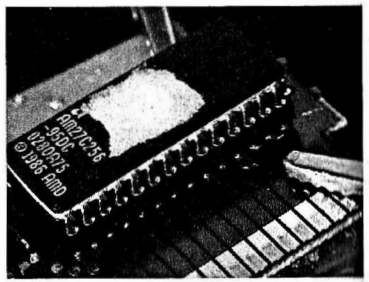
l The next part is the easy part. Get a drill, drill some starholes into the front of the plastic cartridge. Then with your wire cutters you used earlier, bore out the plastic. You need to do this cause the socket we used is too tall for the pastic case of the cartridge, so you need to drill out a hole so the new PRG chip can stick out.

m Don't forget to label it!

h



k



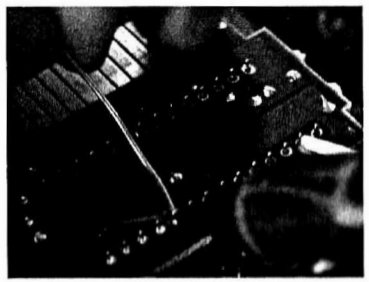
i

l

j

l

j



m