Announcements:

* Quiz 4 on next Monday
  + Will cover new material on mode 4 and this week
* Mode 4 slides updated
* Grades released
* If you are having trouble at this point, talk to the professor

Class: Listed Pros and Cons of Modes 3 and 4

Pros of Modes ¾:

* Mode 4: no flickering
* Mode 3: color range is larger
* Simple to display an image, pixel by pixel
* Bitmap modes easy to learn
  + Just need offset and geometry (basic)

Cons of Modes ¾:

* Mode 3: short window to draw things, flickering
* Mode 4: smaller color range
* No layers, only BG2 used
* Animation is tedious
  + Processer intensive
  + Complete update each frame, need to redo each pixel
    - Constantly redrawing pixels as you go along
* Consumes a lot of memory
  + Takes up most of video memory (use pretty much each pixel)
* Limited to 240x160

We are leaving behind Modes 3 and 4 for now, will return to them. Now we are going into Mode 0.

Demo: Elements of Darkness

* Given limited HW specs, we can still generate awesome visual effects
* Showed us how the background is moving and is far more detailed than current Modes can achieve
* Also showed us a moving character
* In Modes 3 and 4, we constantly have to update and redraw to move things

Tiled backgrounds:

* One such tool that leverages graphics hardware
* Can significantly:
  + Lower the amount of memory an image needs
  + Increase the speed that the image is rendered on screen
* Supported by multiple modes
  + 0, 1, 2
  + We’ll only look at mode 0

Sprites

* Another tool to talk about later on is sprites
* Images that move independently
* Also based on tiles

What is a tile?

* Basic unit of graphics manipulation
  + No more thinking at pixel level, instead in 8x8 chunks
    - 64 pixels
* can be used to generate larger images
* should look at the screen as gridded into 8x8 tiles

Leverage repetition: in the example shown, many images are repeated so you only need to store pattern data once and refer to that pattern multiple times instead of repeating new data

* like color indices, how we reference the color from the index

Means:

* We need to generate a set of unique tiles
  + this is the tile set
  + each element contains the actual color data
* give each tile in the set a number, an index to generate the screen image
* generate backgrounds using a tile map
  + array of indices
  + hope for lots of repetition in the map for reduced space

Example of tiles: there are 16 unique tiles, and the entire image is generated by a tile map where each tile is reused to generate the full image

* too many unique images and you start to resemble bitmap modes, requiring too much memory (so don’t do this)

Mode 3 Tile Maps

* 32x32 tiles
* 256x256 pixels
* 1 map can cover the 240x160 screen
* Backgrounds can be made of large maps, which allows us to scroll through, like when you are moving in a game and the scenery changes
  + Can also be made of multiple maps that we can move through.

Mode 0:

* There’s a map of Mode 0 on the slides
* Currently focusing on how Video Memory Tile Images map to Palette memory with indices

In mode 3 we have the option of making our entries 8 bits or 4 bits

* 4 bits a way of changing efficiency

Palette tradeoff:

* 8 bpp = 256 colors
  + Lots of colors
  + All tiles share the same palette.
  + Tile images take more space
* 4 bpp = 16 palettes of 16 colors apiece
  + Still have 256 colors
    - More constraints
    - More flexibility
  + Tile images smaller (i.e. can have more)
* Steps to tiles:
  + Download or draw graphics
  + Create tile images (USENTI)
  + Create screen image (USENTI)
  + Create a palette (USENTI)
  + Store tile images in a character block
  + Store screen image in one or more
  + screen blocks
  + Store palette in palette memory area
  + Set image control buffer and Mode

Can use Microsoft paint to make tiles, or use more complex software like Photoshop

* Save as a bitmap image
* Use USENTI to create the tile set, tile map and palette
* Use export, save, select tile under Gfx, check Map, check reduce too (reduce can help optimize how much memory you’re using
  + Flip allows you to flip tiles to how you want
* Check palette, hit okay

Using USENTI

* Generated our tile set and tile map

Will have 4096 tiles and entries for indices in our map, and each one of those holds the index, but turns out there is 2 bytes per entry

* Index and the metadata (like horizontally or vertically flip)
* Each screen entry in tile map is more than just an index/number, has extra info in it
* Amount of space taken is really 8192 (4096 x 2)

Cave pictures: reduced to 304 tiles, with 64 pixels per tile (have 304 64-pixel things)

* Each will be an 8 bit value to the palette
* Maps will be how the tiles will be displayed on screen (template, essentially)
* Palette: 256 color palette, each 2 bytes long
* So 256 x 2 = 512

Demo: explore caves with scrolling