LAB 08:

Tiled, Mode 0 Backgrounds & Sprite Animation

Provided Files

- Tiles
    - chipSprite.bmp, chipSprite.pal
    - connectFour.bmp, connectFour.pal
    - connectFourTilemap.tmx, connectFourTiles.tsx
    - tiled-to-gba-export.js
- Example.gba
- game.h
- gba.h, gba.c
- Makefile
- mode0.h, mode0.c
- print.h, print.c
- README.md
- sprites.h, sprites.c

Files to Edit/Add

- .vscode
    - tasks.json
- chipSprite.h, chipSprite.c
- connectFour.h, connectFour.c
- connectFourTiled.h, connectFourTiled.c
- game.c
- main.c

Instructions

In this lab, you will be implementing a sprite that cycles through a simple looping animation. Then, you will create a tiled background in Mode 0 and modify individual tiles at runtime to complete a working version of Connect Four! An `Example.gba` file with the completed lab has been provided for reference, and the controls for it can be found in `README.md`.

TODO 0 – Tiled Installation/Setup

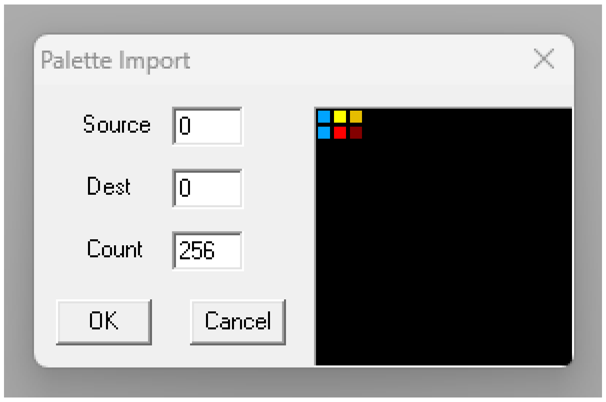
For this lab, you will be using a program called **Tiled** to create and export a background for Mode 0. If you have not done so already, open this Tiled Tutorial document and follow the steps up to and including "Installing GBA Libraries." You **do not** need to follow the steps for setting up a Tiled project as we will already be providing the files you need to modify for this lab.

TODO 1 – Sprite Animation

In this TODO, you will use the provided spritesheet to create a sprite that loops through six frames of animation.

TODO 1.0

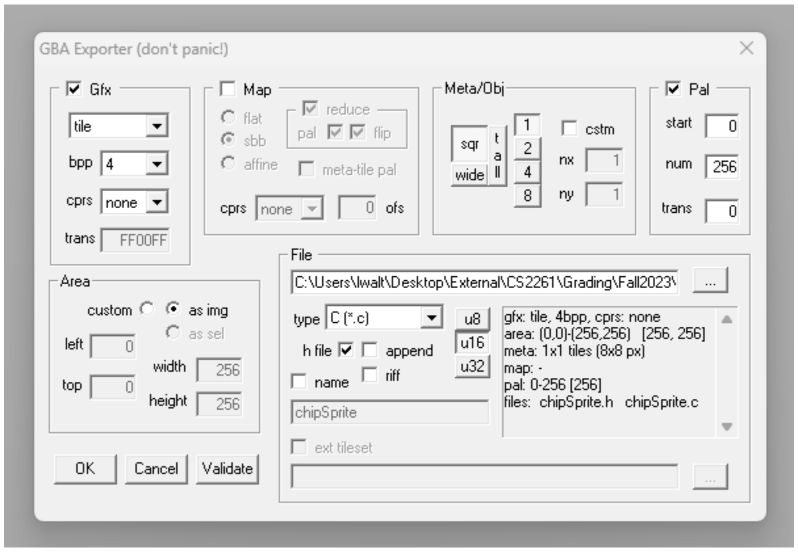
- In Usenti, open `chipSprite.bmp`.
- Import `chipSprite.pal` into the project (**Image** > **Import**, select **.pal** as file type). The palette import screen should look like this:



Palette Import

Source: 0
Dest: 0
Count: 256

- Export the spritesheet as a C source file named `chipSprite` (**Image** > **Export**, select **GBA source file** as file type) using these settings:



GBA Exporter (don't panic!)

Gfx [✓]: tile; bpp: 4; cprs: none; trans: FF00FF

Map [ ]: flat / [✓ sbb] / affine; reduce [✓]: pal [✓], flip [✓]; meta-tile pal [ ]; cprs: none; 0 ofs

Meta/Obj: sqr / wide / tall; 1 / 2 / 4 / 8; cstm [ ]; nx: 1; ny: 1

Pal [✓]: start: 0; num: 256; trans: 0

Area: custom / [✓ as img] / as sel; left: 0; top: 0; width: 256; height: 256

File: C:\Users\lwalt\Desktop\External\CS2261\Grading\Fall2023\

type: C (*.c); h file [✓]; append [ ]; name [ ]; riff [ ]; u8 / u16 / u32

chipSprite

ext tileset [ ]

```
gfx: tile, 4bpp, cprs: none
area: (0,0)-(256,256)  [256, 256]
meta: 1x1 tiles (8x8 px)
map: -
pal: 0-256 [256]
files:  chipSprite.h   chipSprite.c
```

- After you have completed these steps, `chipSprite.h` and `chipSprite.c` should have been created in your project.
  - Make sure that any .c/.h files that you create through Usenti or Tiled are **not** saved in the Tiles folder! The only files that will be compiled into your projects are the ones that are stored **in the same folder as the Makefile**.

TODO 1.1

- At the top of `main.c`, include `chipSprite.h`.

TODO 1.2

- In the `initialize` function of `main.c`, copy the palette and tiles from `chipSprite.h` into the correct memory locations using DMA.
  - Since we're working in Mode 0, which **character block** should the sprite tiles be copied into?
  - Make sure to hide sprites here to prevent cornerface! There is non-transparent color data at tile 0, so it will show up as a tiny square in the top-left corner of the screen by default unless we do this.

TODO 1.3

- In the `initGame` function of `game.c`, initialize the struct members of `chipSprite` that will be used to display different animation frames over time.
  - `timeUntilNextFrame` should be set to 10.
  - `currentFrame` should be set to 0 (starts at the 0th image).
  - `numFrames` should be set to 6 (6 total images in the animation).
  - `hide` should be set to 0 (show sprite at the start of the program).

TODO 1.4

- In the `animateChip` function of `game.c`, set attribute 2 of the chip sprite.
  - The x value of the tile ID will change depending on the current animation frame. Each frame is 2 tiles wide.
  - The palette row should be set to `currentColor->spritePaletteRow`.
    - The chip sprite will switch between red and yellow based on which player is having their turn, and this variable will be set to the palette row needed to display this for you.

TODO 1.5

- If the `hide` member of `chipSprite` is true, hide the sprite in the OAM.
  - Hint: The `|=` operator will be useful in this step!

TODO 1.6

- In the `dropChip` function of `game.c`, set the `hide` member of the `chipSprite` struct to 1 at the start of the function.

- Later in the same function, set `hide` to 0 to make the chip reappear.

TODO 1.7

- In the `animateChip` function of `game.c`, handle the animation logic for switching to the next animation frame.
    - `timeUntilNextFrame` should be decremented every time `animateChip` is called.
    - If `timeUntilNextFrame` is equal to 0:
        - `timeUntilNextFrame` should be reset to 10.
        - `currentFrame` should be incremented to the next frame.
            - What should happen when `currentFrame` reaches the end of the animation?

Build and run. You should see a spinning yellow chip on a black background that moves horizontally whenever LEFT and RIGHT are pressed. If not, fix this before moving on to the next step.

TODO 2 – Mode 0 Background with Usenti & Tiled

In this TODO, you will be using Usenti to import a tileset, and you will use Tiled to build a background tilemap. Make sure that you have completed TODO 0 before completing the following sections.

TODO 2.0

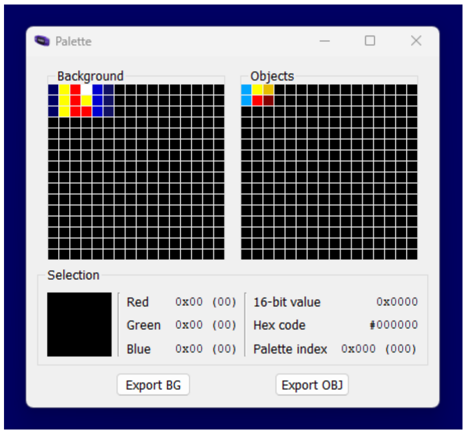
- In Usenti, open `connectFour.bmp` and import `connectFour.pal`.
    - Follow the same steps and settings from TODO 1.0 to import the palette and export the tiles as GBA source files in your project.
    - Name the export `connectFour`.

TODO 2.1

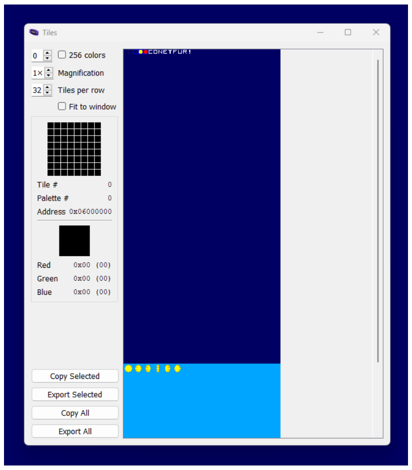
- At the top of `main.c`, include `connectFour.h`.

TODO 2.2

- In the `initialize` function of `main.c`, copy the palette and tiles from `connectFour.h` into the correct memory locations using DMA.
    - The palette should be copied into `BG_PALETTE`.
    - Find where `REG_BG0CNT` is being set. Which **character block** should the tiles be copied to to make them appear on background 0?
- If you build and run the program now, you should be able to see the palette and tiles being stored in your memory! (**Tools** > **View palette** / **Tools** > **View tiles**)



```
Red      0x00  (00)     16-bit value        0x0000
Green    0x00  (00)     Hex code           #000000
Blue     0x00  (00)     Palette index  0x000 (000)
```



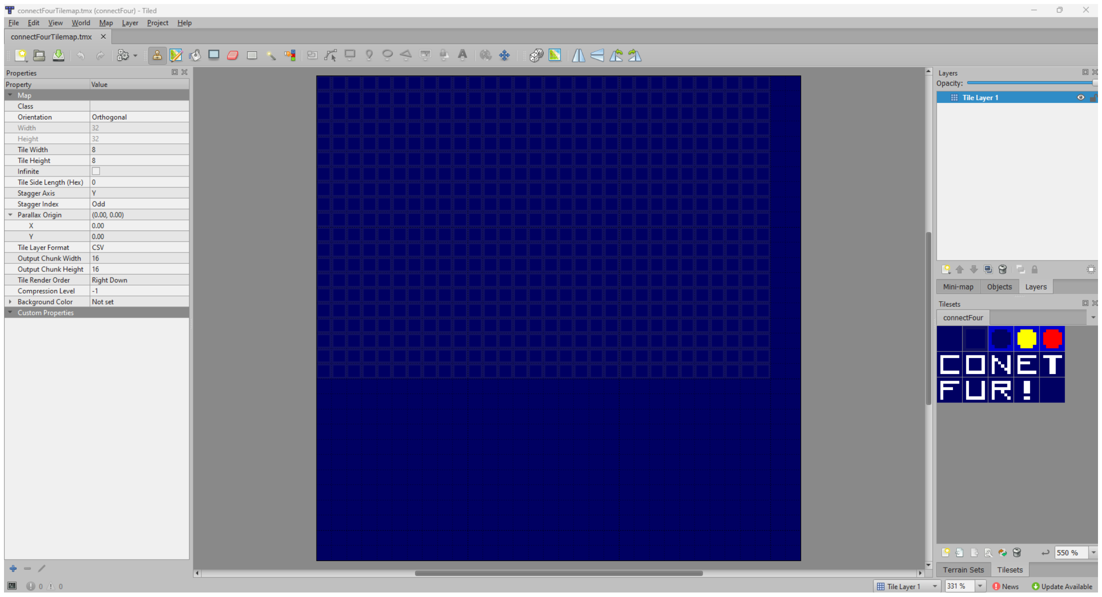
```
0   [ ] 256 colors
1×  Magnification
32  Tiles per row
    [ ] Fit to window

Tile #            0
Palette #         0
Address  0x06000000

Red      0x00  (00)
Green    0x00  (00)
Blue     0x00  (00)
```

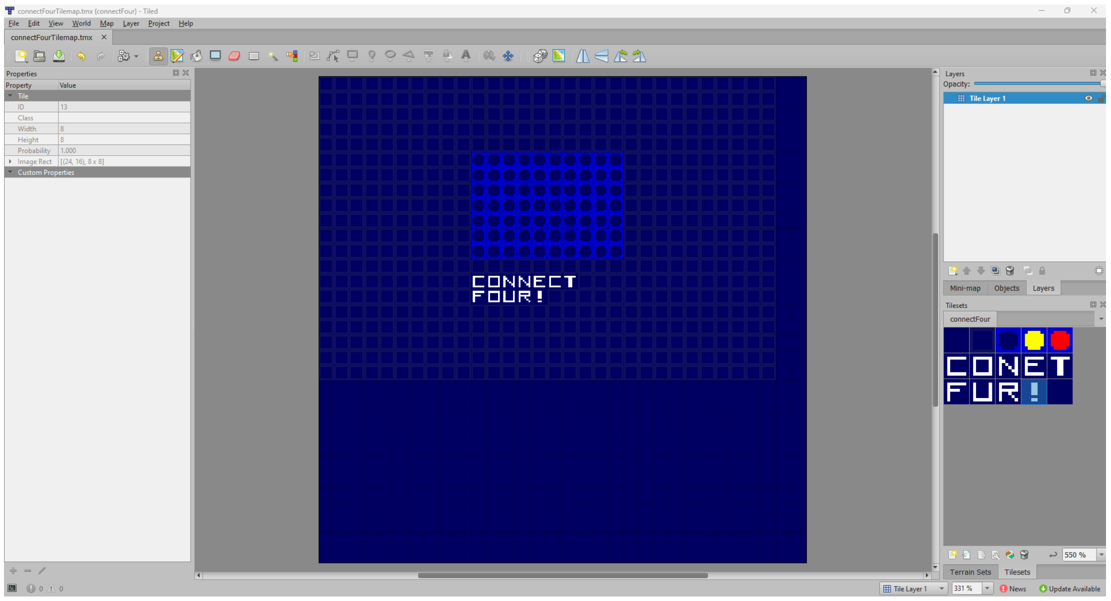
TODO 2.3

- In Tiled, open `connectFourTilemap.tmx`. You should see the following:



Property	Value
Map	
Class	
Orientation	Orthogonal
Width	32
Height	32
Tile Width	8
Tile Height	8
Infinite	☐
Tile Side Length (Hex)	0
Stagger Axis	Y
Stagger Index	Odd
Parallax Origin	(0.00, 0.00)
X	0.00
Y	0.00
Tile Layer Format	CSV
Output Chunk Width	16
Output Chunk Height	16
Tile Render Order	Right Down
Compression Level	-1
Background Color	Not set
Custom Properties	

- The large square in the center of the screen is the tilemap, which determines how the tiles will be arranged on-screen. The tiles in the tileset shown on the right side can be drawn onto the tilemap using the **stamp brush**. Using these tools, create the background shown below:



- For reference, the top-left cell of the grid should be located at (10, 5). The position of the tile your cursor is hovering over is shown in the bottom-left corner of the Tiled application window.

- Export the tilemap as a C source file named `connectFourTiled` (**File** > **Export As**, select **GBA source files** as file type).

TODO 2.4

- At the top of `game.c`, include `connectFourTiled.h`.

TODO 2.5

- In the `initGame` function of `game.c`, copy the tilemap from `connectFour.h` into the correct memory locations using DMA.
  - Find where `REG_BG0CNT` is being set. Which **screen block** should the tilemap be copied to to make it appear on background 0?

Build and run. You should see the tilemap you built displayed on screen. Additionally, pressing the DOWN button will now cause the chip to disappear, and yellow tiles will start to populate the grid in the column you choose (pressing START resets the grid). If you are seeing unexpected behavior with the chip, revisit TODO 1. If there's something wrong with the tilemap, revisit TODO 2.

TODO 3 – Tilemap Modification

In this TODO, you will be writing code to modify your tilemap at runtime. At this point, the DOWN button causes an entire column to be filled with yellow tiles after one chip is dropped, which results in an instant win for yellow. In order to make a functional game of Connect Four, we need to change that!

TODO 3.0

- In `dropChip.c`, use the `TILEMAP_ENTRY_TILEID` macro to remove the yellow tile directly above the one being placed.
    - The tilemap modification code that causes yellow chips to descend down the grid is written just above this TODO. Your answer for this part will be very similar in structure.
    - The tile ID for an empty Connect Four cell is `EMPTY_CELL` (2).

TODO 3.1

- In `dropChip.c`, change the color of the ASCII text (CONNECT FOUR!) when a player wins the game.
- For each tile containing a letter, use the `TILEMAP_ENTRY_PALROW` macro to change the value of the palette row **without altering its tile ID**.
    - Hint 1: Tilemap modification in tiled modes is analogous to setting background pixels in bitmapped modes. This will be a sort of tiled version of the drawRectangle function!
        - The tile containing a "C" (top-left tile for the text) should be located at (10, 13), and the letters should take up a 2x7 area of tiles.
        - The palette row that turns all the letters yellow/red is `currentColor->winTextPaletteRow` (value depends on who placed the last chip).
    - Hint 2: See TODO 1.5!

Build and run. At this point, you should be able to play a fully functional game of Connect Four! Check your submission against the given `Example.gba` before turning in the assignment to ensure that you've completed every step correctly.

Submission Instructions:

Ensure that **cleaning** and building/running your project still gives the expected results. **Please reference the last page of previous assignments for instructions on how to perform a "clean" command.**

Zip up your entire project folder, including all source files, the Makefile, and everything produced during compilation **(including the .gba file)**. Submit this zip on Canvas. Name your submission Lab08_LastnameFirstname, for example:

"Lab08_WexlerHoward.zip"

It is your responsibility to ensure that all the appropriate files have been submitted, and that your submitted zip can be opened and everything cleans, builds, and runs.