LAB 06:
Mode 4

Provided Files

- main.c
- myLib.c
- myLib.h
- game.c
- game.h
- text.c
- text.h
- font.c
- font.h
- katamari.png
- start.png

Files to Edit/Add

- main.c
- myLib.c
- katamari.c
- katamari.h
- start.c
- start.h
- Makefile
- .vscode
    - tasks.json

Instructions

In this lab, you'll be completing several different TODOs, which will piece by piece add Mode 4 drawing to a simple game. Each TODO represents a component of this improvement, and is broken down into sub-TODOs. Your code may not compile until you complete an entire TODO block, at which point the game should compile with the new

drawing function working as expected.

After downloading and unzipping the files, if you take a look at the code, what you see should be familiar. It's a completed Lab05 with a bit of a fun spin! It has a few new features to highlight and test some Mode 4 functionality. If you were to build it right now, however, it will be a blank screen. Complete the TODOs in order, paying close attention to the instructions.

Note: Make sure to copy over your Makefile and .vscode/tasks.json from one of your previous assignments.

TODO 1.0 - SetPixel4
For us to be able to set pixels in Mode 4, we need to account for the additional cases that Mode 4 requires, so we're making a new function.

- In `myLib.c`, complete `setPixel4()`
- Build and run. If you press START, you should be able to travel to the game state and see that the text is drawing correctly (drawChar4 and drawString4 have already been written for you). If not, fix this before going further.
    - The states will be black screens besides the game state (where you should see text drawn)

TODO 2.0 - fillScreen4
We want to be able to see the rest of the states, so we need to fill the screen next.

- TODO 2.0
    - In `myLib.c`, complete `fillScreen4()`
- TODO 2.1
    - At this point, if you compile and run, you won't see anything. That's because our goTo state functions are only filling the screen once (on the hidden page), then doing nothing. To fix this, we need to flip the page with flipPage() after doing our drawing and after we wait for VBlank.
    - In `main.c`, in the `goToStart` function, wait for VBlank, then flip the page.
- TODO 2.2
    - Do the same for `goToPause`
- TODO 2.3:
    - Do the same for `goToWin`
- TODO 2.4:
    - Do the same for `goToLose`
- Build and run your lab thus far. You should be able to travel through all the states you just edited and see their titles printed on them (except for Start because drawFullScreenImage is still blank). If not, fix this before going further.

TODO 3.0 - drawFullscreenImage4

We want to actually be able to see the start screen. Let's make it so we can draw fullscreen images!

- TODO 3.0
  - In `myLib.c`, complete `drawFullScreenImage4()`
- TODO 3.1
  - This isn't worth anything if we don't have an image to draw. Open `start.png` in Usenti.
  - No need to resize this image, as it is already 240x160px (the size of the screen).
- TODO 3.2
  - We are in Mode 4, so the image can only use a MAX of 256 colors. Usenti has a tool to reduce the number of colors.
  - Go to `Palette > Requantize`. Type 256 and hit OK
- TODO 3.3
  - Export your image (`Image > Export`).
  - You will save it as a **GBA source file** in the Lab06 folder.
  - In the export settings, select **bitmap(GBA)** and **"8bpp"** (8 bits per pixel means that it's a char for each pixel, like Mode 4 wants).
  - Also make sure that **Pal** (top right) **is checked**. This will include the palette in the .c file as an array of 256 shorts.
- TODO 3.4
  - `#include start.h` at the top of `main.c`
  - We need to be able to load the image's palette (located in `start.c`) in the game so that the hardware knows what colors to use.
  - In `main.c`, in the `goToStart` function, write a single call to `DMANow()` that will copy the entire start image palette into the game's `PALETTE`
    - You can find the necessary image information declared in `start.h`
- TODO 3.5
  - Uncomment the call to `drawFullScreenImage4` in the `goToStart` function.
- Build and run your lab. You should be able to see all the states now (except for the moving elements in Game). If not, fix this before going further.

TODO 4.0 - drawImage4

We want to be able to draw smaller images as well.

- TODO 4.0

    - In myLib.c, complete `drawImage4()`
- TODO 4.1
    - Open `katamari.png` in Usenti. This one was drawn using a small enough palette and size, so there is no need to resize or requantize.
- TODO 4.2
    - Export this image the same way you did the last one, making sure that Pal is checked.
    - In `initGame()`, write the same `DMANow()` call you wrote to load in the palette last time, this time loading in `katamariPal`.
    - **Note:** this time, after we load in the palette, we also put some of our own colors in there (you can view this in game.c. It looks super complicated, and you don't have to do it this way when you are making your games). It has already been done for you.
- TODO 4.3
    - Include `katamari.h` at the top of `game.c`
    - Uncomment the rest of `drawBall()`
- Build and run. When you reach the game state, you should see one katamari ball bouncing around. If not, fix this before going further.

TODO 5.0 - drawRect4

For our last touch, we need to be able to draw Rectangles

- In `myLib.c`, complete `drawRect4()`
    - This will be by far the longest function you code in this lab. You need to account for ALL cases where you have to set pixels in front or behind each row.
        - **HINT:** draw lots of pictures. There are 4 col/width cases, so make sure you identify and account for all of them.
    - You also need to make sure it still works if you draw a rectangle of small width (1 or 2)
- Build and run.
    - If you shoot, the bullets are now of different widths, and they don't fly straight up when you are moving; they lean in the direction you were moving when you shot. **If any of your homeworks are like this, we will assume you copied code, and penalize heavily.** These additions are to help you test drawRect.
    - The rectangles should all be correctly sized, and they should not wiggle when they move diagonally (they should move smoothly in that direction).

    - When you shoot the rectangles, they should turn into katamari images
- If all this works, submit your lab.

You will know if it runs correctly if:

- You see a fullscreen image on the start state
- When you reach the game state, you see rectangles you are familiar with and the images
- When you shoot the clutter rectangles, they transform into katamari images. Transform all clutter into katamari to win the game.
- If you shoot, a number of different sized bullets show up and you DO NOT see just a white screen when trying to shoot bullets

Tips

- Review lecture and recitation materials for how we deal with Mode 4 pixels
- Review how to use Usenti to export a bitmap image
- Follow each TODO in order, and only move forward if everything is correct

---

Submission Instructions

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 Lab06_FirstameLastname, for example: “Lab06_DamacyKing.zip”.