# LAB 06:
# Mode 4

---

**Provided Files**

- `main.c`
- `gba.c`, `gba.h`
- `mode4.c`, `mode4.h`
- `game.c`, `game.h`
- `font.c`, `font.h`
- `text.c`, `text.h`
- `print.c`, `print.h`
- `imposter.png`
- `start.png`

**Files to Add from Lab00**

- `Makefile`
- `.vscode`
    - `tasks.json`

**Files to Edit/Create**

- `main.c`
- `mode4.c`
- `game.c`
- `imposter.c`
- `imposter.h`
- `start.c`
- `start.h`

---

## Instructions

In this lab, you will complete 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 as expected.

After you download and unzip the files, you should see familiar code. It's a complete Lab04 with a fun spin! It has a few new features to highlight and test some Mode 4 functionality. If you were to build right now, however, it would be a **blank screen**. Complete the TODOs in order, paying close attention to the instructions.

---

## TODO 1: setPixel4

Mode 4 has additional cases that we must consider when setting pixels. We will create a new function to handle these cases.

**TODO 1.0**

- In `mode4.c`, complete `setPixel4()`.

*Build and run*. If you press START, you should be able to travel to the GAME state and see that the score number is being drawn (`drawChar4` and `drawString4` have already been written for you). If not, fix this before going further.

At this point, the states should be black screens besides the GAME state (where you should see text drawn).

## TODO 2: 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 `mode4.c`, complete `fillScreen4()` **using DMA**.

**TODO 2.1**

- At this point, if you compile and run, you shouldn't see anything new in the START state. That's because `goToStart` only fills the screen once (on the hidden page), and then do nothing. To fix this, we need to flip the page with `flipPage()` after doing our drawing, after waiting for vBlank.
- In `main.c`, in `goToStart()`, wait for vBlank, then flip the page.
- **Note:** You still won't see anything since we aren't drawing anything on the start screen yet.

**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*. The `GAME` state should have a light pink background, the `PAUSE` state should have a brown background, and the `LOSE` state should have a salmon background (the `WIN` state should be gold, but for now the game is unplayable without the game objects). If not, fix this before going further.

## TODO 3: `drawFullscreenImage4()`

We want to actually be able to see the `START` screen! Let's make it so we can draw full-screen images.

**TODO 3.0**

- In `mode4.c`, complete `drawFullscreenImage4()`.

**TODO 3.1**

- Having `drawFullscreenImage4()` is worthless if we don't have an image to draw. Open `start.png` in Usenti.
- No need to resize this image, as we already resized it to 240x160px for you (the size of the screen).

**TODO 3.2**

- We are in Mode 4, so our game can only use a **max of 256 colors at a time**. Usenti has a tool to reduce the number of colors.
- To ensure that the image is not going over this limit, go to **Palette > Requantize**. Type 256 and hit OK.
    - This image already has less than 256 colors, but this tool will be important for your future projects!

**TODO 3.3**

- Export your image (**Image > Export**).
- You should save it as a ***GBA source file*** with the name “start” in the lab folder.
- In the export settings, select ***bitmap (GBA)*** and ***8bpp*** (8 bits per pixel means that each pixel is a char, compatible with how the hardware interprets the `videoBuffer` for Mode 4).
- 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 (`#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 `goToStart()`, write a single call to `DMANow()` that will copy the entire start image palette into the palette that the GBA will use for drawing.
    - *Note:* we made a macro for this address, `BG_PALETTE`.
    - You can find the necessary image information declared in `start.h`.

**UNCOMMENT 3.5**

- Uncomment the call to `drawFullscreenImage4()` in `goToStart()`.

*Build and run*. 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: `drawImage4`

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

**TODO 4.0**

- In `mode4.c`, complete `drawImage4()`.

**TODO 4.1**

- Open `imposter.png` in Usenti. This one was drawn using a small enough palette, so there is no need to requantize.
- However, although the picture is of a reasonable size to display on the GBA, images in Mode4 need to have a width that is a multiple of 4.
- Resize your image (**Image > Size**) to be of width 32, and height 32. *Make sure **Stretch** is **not selected***.
    - Make sure that the new pixels added are of the same color as the rest of the background! You can use the pipette tool to select the color from the image, and then the fill/bucket tool to fill that space.

**TODO 4.2**

- Export this image the same way as you did the last one.
- Include `imposter.h` at the top of `game.c`.
- In `initGame()`, write the same `DMANow()` call that you wrote to load in the palette last time, this time loading in `imposterPal`.
    - *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 might look 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.

**UNCOMMENT 4.3**

- Uncomment the code inside of `drawPlayer()`.

*Build and run*. When you reach the GAME state, you should see the player represented by the imposter image. You should be able to press the left and right buttons to move the player. If not, fix this before going further.

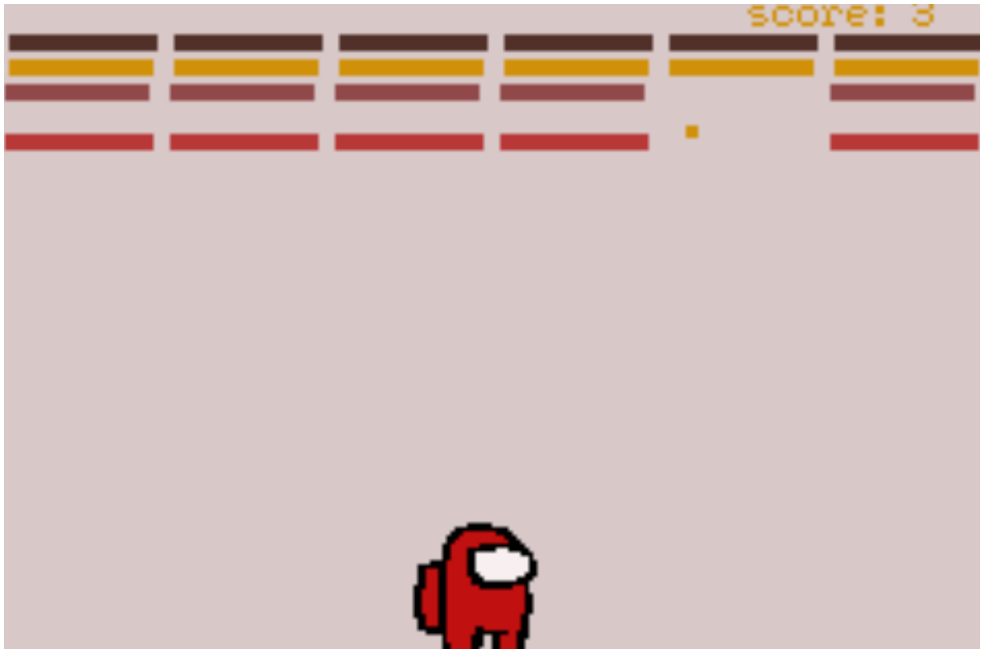
## TODO 5: drawRect4

For our last touch, we need to be able to draw rectangles!

**TODO 5.0**

- In `mode4.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 cases, so make sure you identify and account for all of them.

*Build and run*. The different blocks (rectangles) should not be perfectly aligned. This is to help you see if all 4 cases are working. This addition is to help you test `drawRect4()`. I added comments and mGBA logs to `initBlocks()` to help you see which case might be failing.




This is what a still frame of the GAME state should look like!

---

# Tips

- Review lecture and recitation material 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

Ensure that **cleaning** and building/running your project still gives the expected results. **Please reference the last page of this document 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 Lab06_LastnameFirstname. For example:

> “Lab06_BromanderMarcus.zip”

It is *your* responsibility to ensure that:

- The ZIP file is named correctly
- All the appropriate files have been submitted
- Your submitted ZIP can be opened and everything cleans, builds, and runs