CS 2261
Lab 04: Text and DMA

Provided Files

- main.c
- myLib.c
- myLib.h
- game.c
- game.h
- text.c
- text.h
- font.c
- font.h
- ~~image.png~~

Files to Edit/Add

- main.c
- myLib.c
- game.c
- text.c
- ~~image.c~~
- ~~image.h~~
- ~~Your custom ball image file~~
- ~~Your image's exported .c file (exported using Usenti)~~
- ~~Your image's exported .h file (exported using Usenti)~~
- Your Makefile

Instructions

In this lab, you will be completing several different TODOs, which will, piece by piece, add text, images, and speed to a simple Space Invaders-like 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 improvement.

After you download and unzip the files, add your Makefile, add the SOURCES with it, and then compile and run it. What you end up with should look familiar. It's a completed Lab04! This isn't as great as it could be, though. Complete the TODOs on order, paying close attention to the instructions.

TODO 1 – drawChar

For us to be able to draw text, we need to be able to draw a single character first.

- TODO 1.0: In text.c, complete the drawChar function.
- TODO 1.1: In main.c, in the goToPause() function, use this to draw a capital P.
- Compile and run. You should be able to travel to the Pause state, and see your capital P. If not, fix this before going further.

TODO 2 – drawString

We want to be able to draw entire strings of text with a single function, as well.

- TODO 2.0: In text.c, complete the drawString function.
- TODO 2.1: In main.c, in the goToPause() function, replace your previous drawChar with drawString, and draw "Pause".
- TODO 2.2: In the goToWin() function, draw "Win".
- TODO 2.3: In the goToLose() function, draw "Lose".
- Note: Why do we do this in the goTo functions, and not the every-frame state functions? That's because drawing text takes a long time. We don't want to draw text

every frame unless we absolutely have to.

- Compile and run. You should be able to travel through all the states you just edited and see their titles printed on them. If not, fix this before going further.

TODO 3 – Score

For our game to be user-friendly, we want to be able to see our current progress to victory during the game state.

- TODO 3.0: In the goToGame() function, draw “Balls Remaining: ” in a free area (row 145 and col 5 should be a good spot).
- TODO 3.1: In the game() function, use sprintf() to save the current score (ballsRemaining) in text form to a character array.
    - The char array has already been created for you, called “buffer” sprintf is a function in stdio.h, which we have #included for you It returns null, and works like this:
        - sprintf(arrayName, formatterString, variables, ...)
        - This is exactly like printf, but with an extra argument at the beginning (the arrayName to save to)
        - For more info on sprintf, check https://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm
- TODO 3.2: We want to erase the previous score before we draw the new one. So draw a black rectangle at row 145 col 107 that is the size of a character. Then, draw the score at this location (using drawString with buffer as the character array).
- Note: Why did we draw “Balls Remaining: ” in goToGame(), but the actualnumber in game()? That’s because the “Balls Remaining: ” text does notchange, so we only need to draw it once. The actual score, however, can change at any time, so we have to account for that possibility every frame.
- Compile and run. You should be able to travel to the game state, and see the current score. Shoot some balls, and the score should update. If not, fix this before going further.

TODO 4 – DMANow

For our game to be fast, we need to use DMA. The simplest way to do this is to write a function that sets up the registers of the specified channel.

- TODO 4.0: In myLib.c, complete the DMANow function. This function sets up all the registers of the given DMA channel and turns it on for us. This allows us to use DMA wherever we need it with only a single line, without having to set all of the registers line-by-line in every location we want to use it. There are additional comments in the DMANow function that will help you write it.
- TODO 4.1: Rewrite the fillScreen function to use DMA, using your new DMANow function. Make sure to use DMA channel 3. You may not use any loops.
    - Hint: If you are copying one thing (one source) to every pixel in the videoBuffer, what do you need to tell the DMA control to do (or not do) to the source?
    - Hint: Make sure the src and dst parameters are *addresses* to the locations you are copying from and to.
- TODO 4.2: Rewrite the drawRect function to use DMA, using your new DMANow function. Make sure to use DMA channel 3. You may only use one loop.
    - Hint: You must use one loop here, because unlike fillScreen, the area you are copying to (destination) is not contiguous. Each row of the rectangle is, though. Use DMA to draw one row at a time.
    - Hint: Make sure the src and dst parameters are *addresses* to the locations you are copying from and to.
- Compile and run. The game should look the same, but look a lot snappier during transitions. If this all works, submit your lab. If not, fix it before submitting.

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 Lab04_FirstameLastname, for example:“Lab04_BilboBaggins.zip”.