1 Overview

The goal of this assignment is to make a C program that will run on the GBA emulator. Your program should include everything in the requirements and be written neatly and efficiently. Your main.c should be something different from lecture code, since in this homework you will be creating your own program, but keep the core setup with videoBuffer, MODE 3, waitForVBlank, etc.

Prototypes, #defines, and extern declarations should be put into a myLib.h. It is also optional for you to use other .c/.h files to organize your logic if you wish, just make sure you include them in submission and Makefile.

Please do not make Pong. Everyone asks if they can make Pong, and it's just a boring, low-effort product in general.

1.1 Resources

To tackle this homework, we've provided:

- A myLib.h file that contains all of the necessary GBA declarations such as DMA, videoBuffer, etc.
- A makefile that you can use to compile and run your program by typing make vba

Feel free to use code from these class resources as you need to, but as always, not from your friends or random sketchy internet sites.

1.2 Warnings

- Do not use floats or doubles in your code. Doing so will slow your code down greatly. The ARM7 processor the GBA uses does not have a Floating Point Unit which means floating point operations are slow and are done in software, not hardware. Anywhere you use floats, gcc has to insert assembly code to convert integers to that format. If you do need such things then you should look into fixed point math.
- Only call waitForVBlank once per iteration of your main loop
- Keep your code efficient. If an O(1) solution exists to an algorithm and you are using an O(n2) algorithm then that's bad (for larger values of n)! Contrary to this only worry about efficiency if your program is showing signs of tearing!
- If you use more advanced GBA features like sprites or sound, making them work is your responsibility; we don't really know how they work, so we can't help you.

2 Requirements

- Must be in Mode 3
- You must also implement drawImage3 with DMA. The prototype and explanation are later in the assignment. DMA has not been covered in lecture yet, so you should implement this function with setPixel first, and reimplement it with DMA once it’s been covered in lecture.
- You must use 3 distinct images in your program, all drawn with DMA
  - Two full screened images sized 240 x 160. One of these images should be the first screen displayed when first running your program.
  - A third image which will be used during the course of your program. The width of this image may not be 240 and the height of this image may not be 160.
  - Note: all images should be included in your submission
- You must be able to reset the program to the title screen AT ANY TIME using the select key
- You must create a header (myLib.h), and move any #defines, function prototypes, and typedefs to this file from your code, along with your extern videoBuffer statement if you wish to use videoBuffer in other files. Remember that function and variable definitions should not go in header files, just prototypes and extern variable declarations.
- You must use at least one struct.
- Button input should visibly and clearly affect the flow of the program
- You must have 2-dimensional movement of at least one entity. One entity moving up/down and another moving left/right alone does not count.
- You should implement some form of object collision. For programs where application of this rule is more of a gray area (like Minesweeper), core functionality will take the place of this criteria, such as the numbers for Minesweeper tiles calculated correctly, accurate control, etc.
- You must implement waitforVBlank and the scanlinecounter declaration.
- There must be no tearing in your program. Make your code as efficient as possible!
- Include a readme.txt file with your submission that briefly explains the program and the controls.
- Do not include .c files into other files. Only .h files should be included and .h files should contain no functional code.

3 Deliverables

Please archive all of your source code files as a zip or a tar and upload to Canvas under the Homework 9 assignment.
This includes all .c and .h files needed for your program to compile. Do not submit any compiled files. You can use make clean to remove any compiled files.

Download and test your submission to make sure you submitted the right files.

4 WhattoMake?

Interactive Storybook:

- Recreate a story from a movie or a book using the GBA
- Use text to narrate what is currently happening in the scene
- Use the controls to advance to the next scene or control a character within the scene
- Smooth movement (for any moving characters or objects)
- Start off with a full screen title image and end with a full screen credits image

Galaga:

- Use text to show lives
- Game ends when all lives are lost. Level ends when all aliens are gone.
- Different types of aliens: there should be one type of alien that rushes towards the ship and attacks it
- Smooth movement (aliens and player)

The World’s Hardest Game:

- Smooth motion for enemies and player (no jumping around)
- Constriction to the boundaries of the level
- Enemies moving at different speeds and in different directions
- Sensible, repeating patterns of enemy motion
- Enemies and the Player represented by Structs

Flyswatter:

- Images of yellow jackets or flies moving smoothly across the screen
- Player controlled flyswatter or net to catch the flies
- Score counter to keep track of how many flies have been swatted
- Fullscreen image for title screen and game background
- Enemies and the Player represented by Structs

5 GBACodingGuidelines

5.1 Installing Dependencies

To install dependencies, run

```
$ sudo apt update
$ sudo apt install gcc-arm-none-eabi cs2110-vbam-sdl cs2110-gba-linker-script nin10kit
```

Note that this requires Brandon “The Machine” Whitehead’s CS 2110 PPA, which you should’ve added earlier in the class for complx. If you didn’t (or this is a new VM or something), run the following and then run the two commands above again:

```
$ sudo add-apt-repository ppa:tricksterguy87/ppa-gt-cs2110
```

5.2 Building and Running your Code

To build your code and run the GBA emulator, run

```
$ make vba
```

5.3 Images

As a requirement, you must use at least 3 images in your program and you must draw them all using drawImage3. To use images on the GBA, you will first have to convert them into the suitable format. We recommend using a tool called nin10kit, which you installed with the command above.

You can read about nin10kit in the nin10kit documentation (there are pictures!):

https://github.com/TricksterGuy/nin10kit/raw/master/readme.pdf

nin10kit reads in, converts, and exports image files into C arrays in .c/.h files ready to be copied to the GBA video buffer by your implementation of drawImage3()! It also supports resizing images before they are exported.

You want to use Mode 3 since this assignment requires it, so to convert a picture of smelly festering garbage into GBA pixel format in garbage.c and garbage.h, resizing it to 50 horizontal by 37 vertical pixels, you would run nin10kit like

```
$ nin10kit --mode=3 --resize=50x37 garbage garbage.png
```

This creates a garbage.h file containing

```
extern const unsigned short garbage[1850];
#define GARBAGE_SIZE 3700
#define GARBAGE_LENGTH 1850
#define GARBAGE_WIDTH 50
#define GARBAGE_HEIGHT 37
```

```
#include "garbage.h"
```

OFILES

whichyoucanuseinyourprogrambysaying . The garbage.c generated,

which you should add to the Makefile under as garbage.o if you plan to use it, contains all of the pixel data in a huge array:

```
const unsigned short garbage[1850] =
{
            0x7fff,0x7fff,0x7fff,0x7fff,0x7fff, // ...
            0x7fff,0x7fff,0x7fff,0x7fff,0x7fff, // ...
// ...
            0x7fff,0x7fff,0x7fff,0x7fff,0x7fff, // ...
            0x7fff,0x7fff,0x7fff,0x7fff,0x7fff, // ...

};
```

We've included garbage.png, garbage.c, and garbage.h in the homework zip so you can check them out yourself. To draw the garbage in your own game, you can pass the array, width, height to your drawImage3() like drawImage3(10, 20, GARBAGE WIDTH, GARBAGE HEIGHT, garbage) (to draw at row 10 and column
20). The next section will cover drawImage3() in more detail.

5.4 DMA/drawImage3

In your program, you must use DMA to code drawImage3.

Drawing to the GBA screen follows the same guidelines as the graphics functions you had to implement for HW08. The GBA screen is represented with a short pointer declared as videoBuffer in the myLib.h file. The pointer represents the first pixel in a 240 by 160 screen that has been flattened into a one dimensional array. Each pixel is a short and has a red, green, and blue portion just like the pixels in HW08. With a little modification (Hint: include the graphics and geometry header files and use videoBuffer as the screen buffer), you should be able to use the same code you implemented in HW08 to draw to the GBA screen.

DMA stands for Direct Memory Access and may be used to make your rendering code run much faster.

If you want to read up on DMA before it is covered in lecture, you may read these pages from Tonc. http://www.coranac.com/tonc/text/dma.htm (Up until 14.3.2).

If you want to wait, then you can choose to implement drawImage3 without DMA and then when you learn DMA rewrite it using DMA. Your final answer for drawImage3 must use DMA.

You must not use DMA to do one pixel copies (Doing this defeats the purpose of DMA and is slower than just using setPixel!). Solutions that do this will receive no credit for that function. The prototype and parameters for drawImage3 are as follows.

```
/* drawimage3

* A function that will draw an arbitrary sized image
* onto the screen (with DMA).
* @param r row to draw the image
* @param c column to draw the image
* @param width width of the image
* @param height height of the image
* @param image Pointer to the first element of the image.
*/
void drawImage3 (int r, int c, int width, int height, const u16* image) {
// @todo implement :)
}
```

Protip: if your implementation of this function does not use all the parameters that are passed in then you are not implementing the function correctly. Here is a hint for this function. You should know that DMA acts as a for loop, but it is done in hardware. You should draw each row of the image and let DMA handle drawing a row of the image.

5.5 GBAControls

Here are the inputs from the GameBoy based on the keyboard for the default emulator vbam:

Gameboy	Keyboard
Start	Enter
Select	Backspace
A	Z
B	X
L	A
R	S

The directional arrows are mapped to the same directional arrows on the keyboard.

C Coding Conventions

- Do not jam all your code into one function (i.e. the main function)
- Split your code into multiple files (for example, you can have logic in your main file, library functions in myLib.c with declarations in myLib.h)
- Do not include .c files into other files. Only .h files should be included.
- .h files should contain no functional code.
- Comment your code, and comment what each function does. The quality of your comments will be factored into your grade!