CS 2110 Homework 10
GBA

Henry Harris, Manley Roberts, Gibran Essa

Spring 2019

Contents

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 section 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, you should keep the core setup with videoBuffer, MODE 3, waitForVBlank, etc.

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

Additionally, we want to make one point very clear: please do not rehash lecture code in your program. This means that you are not allowed to just slightly modify lecture code and to call it a day. If we open your program and we see several boxes flying in random directions, that will be a very bad sign, and you will not receive a very pleasant grade.

1.1 Resources

To tackle this homework, we've provided:

- A gba.h file that contains all of the necessary GBA declarations such as DMA, videoBuffer, etc.
- Several other files which contain more "starter" code to get you rolling. See Section 5.7 for details.
- A makefile that you can use to compile and run your program by typing `make vba`

Feel free to use code from 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 as they 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 that you think requires floats or doubles, 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 $O(n^2)$ 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 choose to use more advanced GBA features like sprites or sound, making them work is your responsibility; we (the TAs) do not really know how they work, so we sadly can't help you.

2 Requirements

There is no autograder for this assignment, so your grade entirely depends on the following requirements. These requirements will be checked by a TA during your demo, so be sure to cover them all!

- Your program must be in Mode 3
- You must also implement drawImage3 with DMA. The prototype and explanation are later in the assignment. Depending on when you are reading this, DMA may not have been covered in lecture yet. If this is the case, 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 240x160. One of these images should be the first screen displayed when launching your program.
  - A third image which will be used during the course of your program. The width of this image must be less than 240 pixels and the height of this image must be less than 160 pixels.
  - 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. This resets the ENTIRE program, including game state
- 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. Using the suggested `AppState` is a good way to fulfill this.
- Button input should visibly and clearly affect the flow of the program
- You must have 2-dimensional movement of at least one entity (an entity that moves both left/right **and** up/down). 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 can take the place of this criteria, such as the numbers for Minesweeper tiles calculated correctly, accurate control, etc. When in doubt, ask a TA for clarification.
- You must implement `waitforVBlank()` and the `scanlinecounter` declaration.
- Use text to show progression in your program. Use the example files from lecture as reference. You can also find more information in Tonc: `http://www.coranac.com/tonc/text/text.htm`
- 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.
- As always, 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 Gradescope under the `Homework 10` assignment. This includes all .c and .h files needed for your program to compile and run. Do not submit any compiled files. You can use `make clean` to remove any compiled files, or `make submit` to remove compiled files *and* create the tar archive!

All non-compiling homeworks will receive a zero. If you want to avoid this, be sure to use the Makefile as described.

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

4 What to Make?

You may either create your own program the way you wish it to be as long as it fulfills the requirements, or you can make programs that have been made before with you own code. However, your assignment must be yours entirely and not based on anyone else's code. This also means that you are not allowed to base your program off the code posted from lecture. Programs that are lecture code that have been slightly modified are subject to heavy penalties. Below are some previous programs that you can create or use as inspiration:

4.1 Example Programs

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
- Characters represented by structs

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)
- Aliens and the ship represented by structs

The World's Hardest Game:

- Smooth motion for enemies and player (no jumping around)
- Confined 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 flies moving smoothly across the screen
- Player controlled flyswatter to swat 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 GBA Coding Guidelines

5.1 Installing Dependencies

If you are not using Docker, you will have to install some dependencies before you are able to begin using GBA. Run the following commands.

```
$ sudo apt update
$ sudo apt install gcc-arm-none-eabi cs2110-vbam-sdl mednafen 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/dual boot 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 VBA emulator, run

```
$ make emu
```

5.3 Images

As a requirement, you must use at least 3 images in your program. To use images in GBA, you will first have to convert them into the suitable format. We recommend using a tool called `nin10kit`, which is pre-installed on the Docker image, or you just installed using 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
```

which you can use in your program by saying `#include "garbage.h"`.

The `garbage.c` generated, which you should add to the Makefile under `OFILES` 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.

The GBA screen is represented with a `short` pointer declared as `videoBuffer` in the `gba.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 red, green, and blue channels.

DMA stands for Direct Memory Access and should 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 actually 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 start drawing the image at
* @param c column to start drawing the image at
* @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)
{
    // TA-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. You should know that DMA acts as a for loop, but it is done in hardware.

5.5 GBA Controls

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.

5.6 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.

5.7 Making Sense of the Files

As mentioned in the C Coding Conventions section, it's often a good idea to split up functionality into multiple files. In fact, this is exactly what we've done with the "starter" code we've given you. The pure volume of files may be a bit daunting, so here's a brief breakdown of what each file is used for.

- `Makefile`

  This Makefile contains all of the tasks you can run to build and test your program. You'll need to modify it to include the `.o` file for any image you'd like to use. However, you should not modify the bottom portion of this file, as bad things may happen. Feel free to look through it in order to see all of the tasks at your disposal.

- `main.c`

  This file contains a state machine which ultimately calls all other functionality in the program. It is also the main entry point to the entire application.

- `logic.h`

  This file contains the declaration of `AppState`, the struct which will contain whatever persistent information your app requires in order to process change/motion/progress/etc. This file should not contain references to graphics code.

  This file also contains some prototypes for functions in `logic.c`.

- `logic.c`

  This file contains the functions used to initialize and process the `AppState` struct which is responsible for tracking the current state of the program.

- `draw.h`

  This file contains some prototypes for functions in `logic.h`. It depends on logic.h because draw.c has functions which take in `AppState` structs, which are declared in `logic.h`.

- `draw.c`

  This file will contain functions which define the rules for drawing either the entire possible screen, based on a given application state, or drawing and undrawing very small portions of it in order to reflect frame-to-frame changes. It's likely that these functions will actually refer to other functions in `gba.c`

- `gba.h`

  This file contains a large collection of useful macros and constants which will help primarily with GBA-specific tasks. These include macros for handling GBA input, DMA graphics, and general GBA graphics.

  The file also contains an extern declaration of the font data found in `font.c`, which is necessary for drawing text.

  This file also contains some prototypes for functions in `gba.c`.

- `gba.c`

  If `draw.c` defines the high-level rules for what graphics should be drawn, the functions you will write in this file do the "dirty" work, executing graphics updates with both DMA and non-DMA strategies. All of this code will be very specific to the GBA platform and the way it handles graphics.

  The file also comes with some prepackaged functions for drawing text.

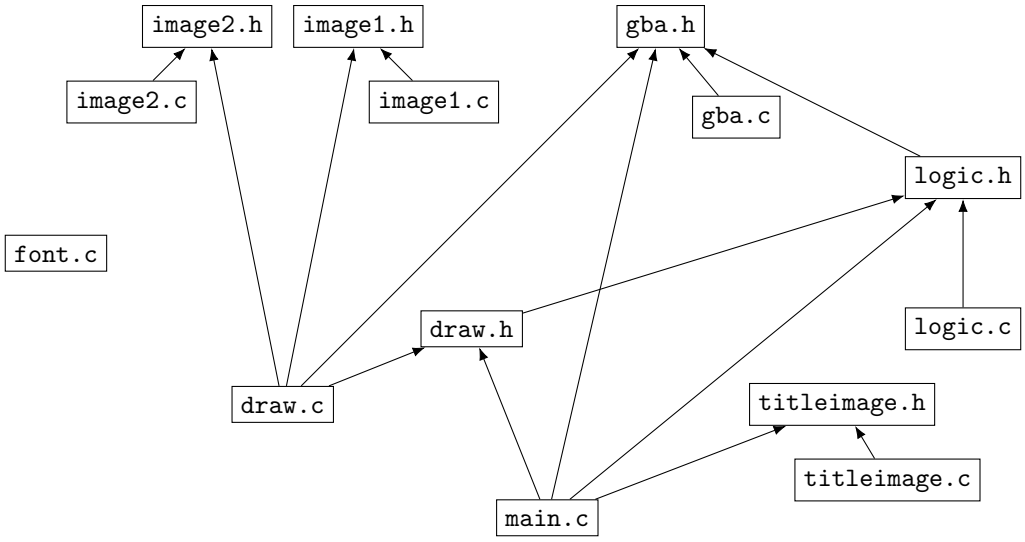
- `font.c`

  Simply exists to store a large amount of font data. No real need to mess around with this file.

- Various Image Files

  When you create your own image files, you will need to include the relevant header files in any file you'd like to reference these images from. The data stored in each of these files (and how to create them) is pretty well explained in section 5.3.

Here's a sample chart of what your `#include` dependency tree might look like.



5.8 Submit

To submit your code:

1. Make sure your code compiles by running `make emu`
2. Clean the code by running `make clean`
3. Create the submission tar by running `make submit`
4. Turn in submission.tar.gz on gradescope!

6 Rules and Regulations

6.1 General Rules

1. Starting with the assembly homeworks, any code you write must be meaningfully commented. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.
2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.
3. Please read the assignment in its entirety before asking questions.
4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

6.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.

2. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.

3. Do not submit compiled files, that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

6.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM. You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas/Gradescope, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

6.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply

an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

6.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own. Consider instead using a screen-sharing collaboration app, such as `http://webex.gatech.edu/`, to help someone with debugging if you're not in the same room.

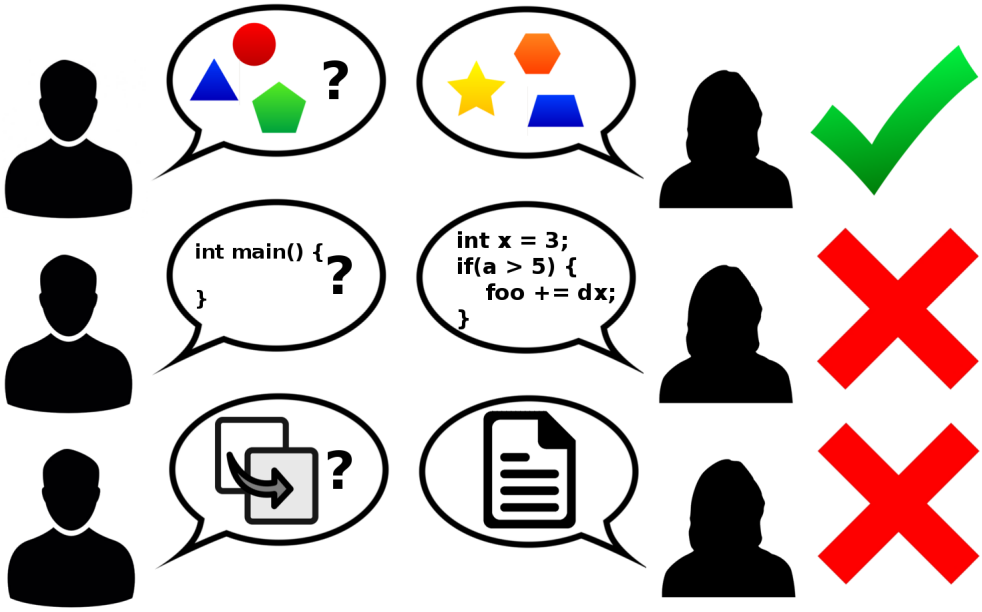


Figure 1: Collaboration rules, explained colorfully