CS 2110 Homework 10
GBA

Cem Gokmen, Henry Harris, Austin Adams

Fall 2018

Contents

1 Overview

The goal of this assignment is to make an interactive C program (e.g. a game, an interactive storybook, an application of other sorts) that will run on the GBA emulator. Your program should include everything in the requirements and be written neatly and efficiently. Your code 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 header files such as logic.h, graphics.h, or gba.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.

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

2 Assignment Details

IMPORTANT! READ THIS SECTION VERY CAREFULLY.

2.1 Resources

To help you structure your project, we have provided you with a number of files as a template. These files contain comments marked **TA-TODO** for things that need to be worked on. Some of these are things that you **MUST** have in your submission as explained in the requirements section below, such as waitForVBlank. Others are things that we have included to guide you to intelligently structure your project. As a result, before starting with your project, **make sure you search in the entire directory for the string TA-TODO to find all the things we recommend that you do.**

The structure we give you is as follows:

- *main.c* contains the state machine behind the application that controls what screen you are in, waits for VBlank, and calls all other functions.
- *gba.c/h* contain the GBA macros and the low-level drawing functions, as well as a random number generator that you can use.
- *graphics.c/h* contain prototypes for functions that you will write that will draw your app on the screen.
- *logic.c/h* contain structs and function prototypes for the logic layer behind your app. The idea here is that we keep the logic and graphics layers separate: the app, with all its relevant information, is entirely maintained as a struct while the graphics layer in *graphics* is responsible for representing this logical app on the screen.
- *font.c* contains the font that the text-drawing functions use. You can swap this out with any font you want to use.
- *images/* is a good place to put the .c and .h files corresponding to images you want to use, produced by nin10kit.

Some ideas that we put in the structure that we recommend you implement are as follows:

- A VBlank Counter: in *gba.c/h* we declare and initialize a vblankCounter variable that you will increment every time we wait for VBlank. Then you will be able to use this variable from inside your *processAppState* etc. functions to do something once every fixed number of frames, for example, move a snake every 10 frames by checking if *vblankCounter % 10 == 0*.
- Key Just Pressed macro: in *gba.c/h* we give you a Key Just Pressed macro that you can use to determine if the key has been recently pressed, i.e. it wasn't 'down' before but it is now. You need to implement this macro (it's essentially this previous sentence, in C, using the *KEY_DOWN* macro). We keep track of the previous and current key inputs in *main.c* and pass those to *processAppState* for you to use this macro with.
- Current and Next AppState: in *main.c* we give you two structs in which we maintain the app state: currentAppState and nextAppState. The purpose of this is to enable the calculation of everything about the next state without having to wait for vblank, but then doing the drawing during VBlank. We need the previous state for exactly this reason: even after we have calculated the next state, we need to be able to access the previous state to be able to undraw (i.e. erase) the elements in the previous state that might have moved. In fact, you can use this setup to compare elements between the two states so that you only draw what has changed.

However, the structure that we give you is only a suggestion. You do NOT need to use any of the files we give you, in fact, you do NOT need to use anything we give you, as long as your code satisfies the requirements and the command *make* produces a .gba file. Feel free to manipulate the template we give you in absolutely any way you want.

2.2 Content Requirements

- 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
- 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.
- Use text to show progression in your program.
- There must be no tearing in your program. Make your code as efficient as possible!

2.3 Code Style Requirements

- Must be in Mode 3 unless you're absolutely sure you want to try something else, without TA support.
- You must also implement a drawImage function with DMA.

- Put your prototypes, struct declarations and typedefs in header files. Do not put functions in header files.
- You must use at least one struct.
- 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.

While full credit will be given for submissions that satisfy all the above requirements, extra credit for submissions that go above and beyond may be given at TA and instructor discretion.

2.4 What to Make?

You may either create your own program the way you wish it to be as long as it covers 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. Here are some previous programs that you can either create or use as inspiration:

Hint: It's better to be creative and come up with something new than reuse these!

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

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
- Satisfy the collision requirement by having a cursor etc. element whose position needs to be used to calculate where it's pointing.
- Smooth movement (for any moving characters or objects)
- Start off with a full screen title image and end with a full screen credits image

Data Visualization App:

- Use the GBA to visualize a dataset that you can include in your code. Can be a 2d plot.
- Use text to draw the scale of the axes.
- Use a cursor and apply collision calculation to click and zoom on individual datapoints.
- Implement zooming etc. animations without tearing.

3 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 $O(n^2)$ algorithm then that's bad (for larger values of n)! For example, **do not pass large structs directly to functions** but pass pointers instead so that only a single pointer needs to be copied onto the stack rather than the full struct.
- If you use more advanced GBA features like sprites or sound, making them work is your responsibility; we (the TAs) can not help you.

4 Deliverables

Please run `make submit` and upload `submission.tar.gz` to Gradescope under the `Homework 10` 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. (`make submit` will do this automatically.)

Gradescope will confirm that your Makefile produces a valid .gba file when `make` is run.

Download and test your submission to make sure you submitted the right files. The Gradescope tester just checks that you submitted compiling code — not that you submitted what you thought you did.

5 GBA Coding Guidelines

5.1 Installing Dependencies

There are some dependencies for running the tests.

If you are using Docker, re-run `cs2110docker.sh` to stop the old container, download updates to the image (which include these dependencies), and restart the container.

If you are using a VM, run

```
$ 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 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 emu
```

5.3 Images

As a requirement, you must use at least 3 images in your program and you must draw them all. 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 `drawImage`! 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

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

This creates a `garbage.h` file in `images/` 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 "images/garbage.h"`.

The `images/garbage.c` generated, which you should add to the Makefile under `OFILES` as `images/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 `images/` directory in the homework tarball so you can check them out yourself. To draw the garbage in your own game, you can pass the array, width, height to your `drawImage` like `drawImage(10, 20, GARBAGE_WIDTH, GARBAGE_HEIGHT, garbage)` (to draw at row 10 and column 20). The next section will cover `drawImage` in more detail.

5.4 DMA / drawImage

In your program, you must use DMA to implement image drawing.

Drawing to the GBA screen follows the same guidelines as the graphics functions you had to implement for HW9. 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 a red, green, and blue portion just like the pixels in HW9. 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 HW9 to draw to the GBA screen. Note that those functions do not use DMA and thus are considerably slower than the ones you will implement for this assignment.

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 drawImage without DMA and then when you learn DMA rewrite it using DMA. Your final answer for drawImage 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. When drawing an image, you should know that DMA acts as a for loop, but it is done in hardware. You should call DMA for each row of the image and let DMA handle drawing all of the columns in that row of the image.

For drawing a fullscreen image, a single DMA call suffices. Why? Try implementing this in your code.

5.5 GBA Controls

Here are the inputs from the GameBoy based on the keyboard we've configured for the recommended emulator mednafen (these are the same as the default controls for vbam, an alternate emulator):

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)
- 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!

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.

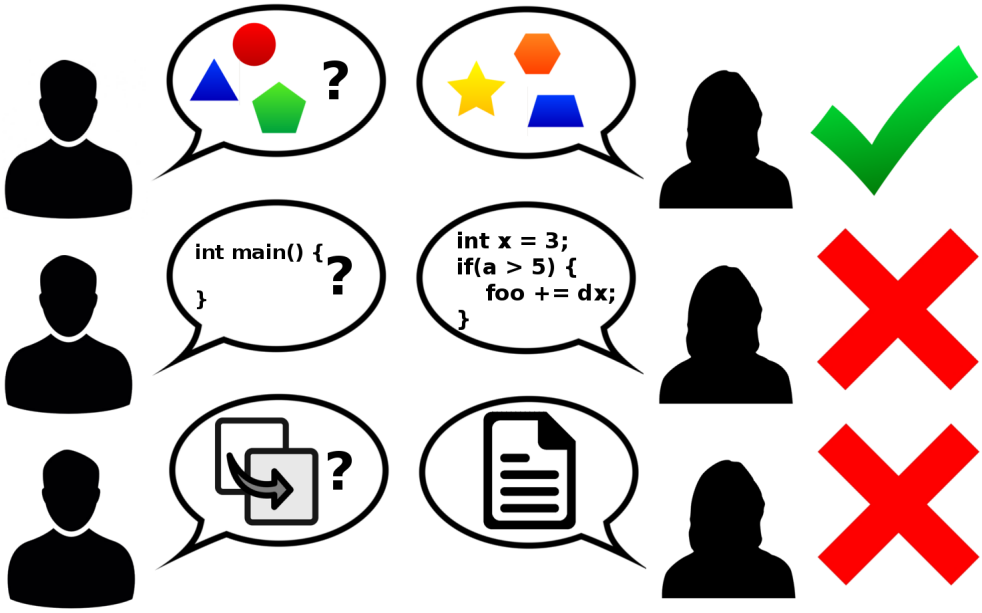

Figure 1: Collaboration rules, explained colorfully