CS2110 Fall 2015
Homework 09

This assignment is due by:

Day: Wednesday October 28th, 2015
Time: 11:54:59pm

Rules and Regulations

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 T-Square. When you submit the assignment you should get an email from T-Square telling you that you submitted the assignment. If you do not get this email that means that you did not complete the submission process correctly. 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 T-Square.

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 11:54PM.* You alone are responsible for submitting your homework before the grace period begins or ends; neither T-Square, 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.

General Rules

1. Starting with the assembly homeworks, Any code you write (if any) must be clearly commented and the comments must be meaningful. You should comment your code in terms of the algorithm you are implementing we all know what the 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.

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 T-Square or you may submit an archive (zip or tar.gz only please) of the files (preferred). You can create an archive by right clicking on files and selecting the appropriate compress option on your system.
3. If you choose to submit an archive please don't zip up a folder with the files, only submit an archive of the files we want. (See **Deliverables**).
4. 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.
5. Do not submit links to files. We will not grade assignments submitted this way as it is easy to change the files after the submission period ends.

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 electronic 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. 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, etc.

Overview

The goal of this assignment is to get you familiar with the GBA environment and to write your first GameBoy game! You will need to create several files that we ask for, and you will also need to implement several required functions. Your game should include everything in the **Requirements** section and be written neatly and efficiently.

You may either pick one of the games from our list, or you may create your own that is of the same or greater complexity as those options. If you are unsure about whether or not your game idea fits the requirements, feel free to check with the TAs.

Additionally, we want to make one point very clear: **please do not rehash lecture code in your game**. This means that you are not allowed to just slightly modify lecture code and to call it a day. If we open your game 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.

Part 1

It's time to create a library file for your GBA game. You will be using sets of these files to create your games for the next few assignments. Please keep the following in mind: We are going to create two .c files: a `main.c` and a `mylib.c`. The purpose of the `main.c` file is to write all of your game logic. For instance, you may have a neat function that will initialize all of the bricks' locations for a Breakout game; this would go in your `main.c` file. Other useful functions that would apply to any game you could write, for example, a function that draws a rectangle on the screen would go in `mylib.c`. Please keep your files organized, as it will make your TAs happy.

In addition, if you do not understand a concept, then you should be reading the required C book for this class. We will be moving quickly through this stuff and if you fall behind, your grade will hurt. Terms you should know by the end of this assignment are italicized.

We will be creating the following files: `mylib.c` and `main.c`

`mylib.c`

Please add the following pre-processor declarations to this file:

- **u16** – This will be an alias of the unsigned short type. You don't want to type `unsigned short` all of the time so you should use *typedef* to make u16 an alias of unsigned short, e.g. you could write `u16* videoBuffer` instead of typing out `unsigned short* videoBuffer`.
- **videoBuffer** - A **global variable** pointing to the start of video memory. This is located at 0x6000000 and should **point** to a u16.
- ***Function prototypes*** – for each function you have to declare in `mylib.c`. These should be located above the actual function implementations. These are forward declarations of functions without defining them, so the compiler knows their signature ahead of time. You will see in Part 2 why this is useful.
- ***Functions*** – You need to implement these four functions. (You may switch row and column with x and y. I personally prefer (x, y), Bill likes (r, c)).

```
// A function to set pixel (r, c) to the color passed in.
void setPixel(int r, int c, u16 color)
{
    // @todo implement :)
}
```

```
// A function to draw a FILLED rectangle starting at (r, c)
void drawRect(int r, int c, int width, int height, u16
color)
{
                // @todo implement :)
}


// A function to draw a HOLLOW rectangle starting at (r,
c)
// NOTE: It has to run in O(w+h) time.
void drawHollowRect(int r, int c, int width, int height,
u16 color)
{
// @todo implement :)
}
```

drawimage3

You must implement this function inside of your library file to draw an image of an arbitrary size and at an arbitrary location on the screen. The prototype and parameters for drawImage3 are as follows.

```
/* drawimage3
 * A function that will draw an arbitrary sized image
 * onto the screen
 * @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 :)
}
```

This function should be very similar to the one that you implemented in the previous homework. You are essentially copying pixel values from the image array into the GBA's memory. The position of the data in memory determines where the image is drawn on the screen.

Part 2

`main.c`

For `main.c`, we will need to include a few declarations as well. Notice that some of the following declarations are duplicates from the `mylib.c` file. In your next assignment, we will be required to use header (.h) files which will allow you to remove much of this redundancy. If you know how to use header files (from the previous homework), feel free to go ahead and implement a mylib.h file, but this is not required for the assignment.

- **REG_DISPCNT** - The display control register, located at 0x4000000. This will be a ***symbol*** that will **access** the memory location 0x4000000 and get the **unsigned short** located at 0x4000000 for us.
- **RGB** - A *macro* you should `#define` at the beginning of your file. This macro takes three integers representing the red, green, and blue components of the color and returns the corresponding color value. Your macro should work in all cases e.g. RGB(2+2, 4+4, g+2) and should respect order of operations, based on the examples from class! (Hint: "Encapsulation")
- **u16** – This will be an alias of the unsigned short type. You don't want to type unsigned short all of the time so you should use *typedef* to make u16 an alias of unsigned short.
- **videoBuffer** - The global variable videoBuffer declared in `mylib.c` should be visible in any file that needs to use it. To do this, extend the scope of videoBuffer to main.c by using the `extern` keyword when referring to the variable from inside main.c.
- ***Function prototypes*** – for each function you had to declare in `mylib.c.` This is so the compiler knows how to link between `mylib.c` and `main.c`.
- **A main function as defined below**:

```
int main(void)
{
      // Put your code here
}
```

In your main function you should set up REG_DISPCNT as shown in class.
Remember that we want mode 3 and to enable background 2.

You will then have a busy loop somewhere in your code to keep the program from exiting. For example, you could use while(1). You would set up your drawing inside of the while-loop.

Your Goal

After implementing the functions above, you will have a toolkit for drawing to the screen and making your game. You will be using these function and any additional functions that you wish to implement. Feel free to implement functions to draw other shapes, like triangles or circles!

For the game, we are not expecting amazing works of art from you, but at the same time we are expecting a bit more than just random boxes placed around the screen for no reason. **A part of the grading criteria is dedicated to the creativity and the demonstrated work ethic of your submission**, so please do not try to rush through the assignment. You are much more likely to receive a great grade from your TA if it is clear that you have put in some time and thought into your homework.

This is one of the most open-ended assignments of this course. The GBA screen is your canvas, and there are many different ways that you could go. That said, you must follow the rules listed below in order to not lose major points. Remember that the work that you produce should be substantially different from the examples shown in lecture and from the works of others.

Requirements

1. Must be in Mode 3
2. Images – You must use 3 (or more) distinct images in your game
   a. A title screen sized 240x160
   b. A game over screen sized 240x160
   c. A third image which will be used during game play. The width of this image may not be 240 and the height of this image may not be 160. In other words, it should not occupy the whole screen.
   d. Note: all images should be included in your submission
3. You must implement the 4 required library functions, including **drawImage3** that is able to draw images of arbitrary sizes at arbitrary locations.
4. You must be able to reset the game to the title screen AT ANY TIME using the select key
5. You must use at least one **struct**. This could be a player or an enemy.
6. **Button input** should visibly and clearly affect the flow of the game
7. You must have **2-dimensional movement** of at least one entity.
8. You should implement some form of object collision that works consistently.
9. You must implement **waitforVBlank** and the `scanlinecounter` declaration. **There must be no tearing in your game. Make your code as efficient as possible!**
10. Button input (standard GBA buttons) that clearly affects the flow of the game.

You **may, but are not required to,** create a header file (**myLib.h**), and move any #defines, function prototypes, and typedefs to this file from myLib.c, along with your extern videoBuffer statement. Remember that function and variable definitions should not go in header files, just prototypes and extern variable declarations.

Include a **readme.txt** file with your submission that briefly explains the game and the controls.

Images

As a requirement you must use at least 3 images in your game and you must draw them all using drawImage3. We have provided a tool that will make this task easier for you

CS2110ImageTools.jar – available on T-Square in Resources/GBA

Similarly to your previous homework, CS2110ImageTools.jar reads in, converts, and exports an image file into a format the gba can read – .c/.h files! It also supports resizing the images before they are exported.

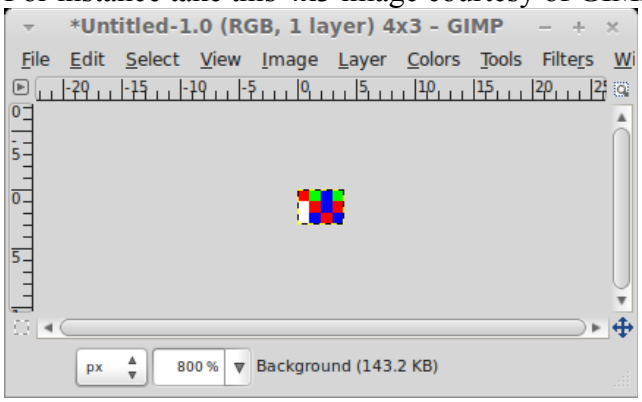
You may run the program from terminal in the directory where you downloaded it like this:

```
java -jar CS2110ImageTools.jar
```

CS2110ImageTools.jar will give you a graphical interface to load image files and save their converted files. The output of this program will be a .c file and a .h file. In your game you will #include the header file. It contains an extern statement so any file that includes it will be able to see the declarations given in the .c file CS2110ImageTools.jar exported. Inside the exported .c file is a 1D array of colors which you can use to draw to the screen.

Example

For instance take this 4x3 image courtesy of GIMP



When this file is exported here is what the array will look like

```
const unsigned short example[12] =
{
      // first row red, green, blue, green
      0x001f, 0x03e0, 0x7c00, 0x03e0,
      // white, red, blue, red
      0x7fff,0x001f, 0x7c00, 0x001f,
      //white, blue, red, blue
      0x7fff, 0x7c00, 0x001f, 0x7c00,
}
```

The number of entries in this 1D array is 12 which is 4 times 3. Each row from the image is stored right after the other. So if you wanted to access coordinate (row = 1, col = 3) then you should get the value at index 7 from this array which is red.

Button Inputs

Your game must use button inputs – otherwise it would not be a game! Refer to the lecture slides and to these button inputs for the emulator:

GameBoy	Keyboard
Start	Enter
Select	Backspace
A	Z
B	X
L	A
R	S
Left	Left Arrow
Right	Right Arrow
Up	Up Arrow
Down	Down Arrow

Holding the space bar will make the emulator run faster. This might be useful in testing, but the player should never have to hold down space bar for the game to run properly since there is no space bar on the actual GBA.

You can learn more about button inputs on this site: http://www.coranac.com/tonc/text/keys.htm

Warning

Two things to note for now

1. 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. If you do need such things then you should look into fixed point math (google search).
2. Do not do anything very intense; you can only do so much before the GBA can't update the screen fast enough. You will be fixing this problem in the next assignment.

I **STRONGLY ADVISE** that you go **ABOVE AND BEYOND** on this homework. **PLEASE do not just sprinkle your screen in boxes** – such submissions will **lose** points. Draw something amazing, or implement something really cool for the gameplay! R**emember that your homework will be partially graded on its creative properties and work ethic**. You are much more likely to make your TAs very happy and to have a better demo/grade if you go above and beyond what is required.

You may research the GBA Hardware on your own. You can read tonc (available at http://www.coranac.com/tonc/text/), however you may not copy code wholesale from this site. The author assumes you are using his GBA libraries, which you are not.

If you want to add randomness to your game then look up the function rand() in the Man pages. Type “man 3 rand” in a terminal.

What Game to Make?

You may either create your own game the way you wish it to be as long as it covers the requirements, or you can make games 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 game off the code posted from lecture. Games that use Bill's code that have been slightly modified are subject to heavy penalties.** Here are some previous games that you can either create or use as inspiration:

Galaga: http://en.wikipedia.org/wiki/Galaga
Requirements:
1. Accurate, Efficient O(1) Rectangular Collision Detection implemented as a function.
2. Lives, you can use text or images to show them.
3. Game ends when all lives are lost. Level ends when all aliens are gone.
4. Different types of aliens – there should be one type of alien that rushes towards the ship and attacks it
5. Smooth movement (aliens and player)

The World's Hardest Game (Challenge our skill with your game):
http://www.addictinggames.com/action-games/theworldshardestgame.jsp
Requirements:
1. Accurate, Efficient O(1) Rectangle Collision detection as a function.
2. Smooth motion for enemies and player (no jumping around)
3. Constriction to the boundaries of the level.
4. Enemies moving at different speeds and in different directions.
6. Sensible, repeating patterns of enemy motion
7. Enemies and the Player represented by Structs

Frogger: http://en.wikipedia.org/wiki/Frogger
Requirements:
1. The Frog and the logs/lily pads must be represented by Structs internally.
2. O(1) collision detection with any object. If the frog collides with traffic, it dies. If it does not land on a lilypad/log, then it also dies. The materials in the river "lily pads/logs" must move the froggy along with them.
3. There is a time limit in which the frog must get to his home if time expires then the frog also dies (hint there are about 60 vblanks per second.)
4. Once a Froggy occupies a home, another frog cannot occupy that home.
5. The player must have a set amount of lives they can lose before losing. The number of lives must be displayed to the user. The game is over when all of the lives are lost. The game is won if all frogs get to their homes.

These are just some suggestions to get you on the right track with making your game. If you are having any trouble deciding or an idea you would like to check, please consult with the TAs.

Part 3 - Makefile

The makefile is a file used by make to help build your gba file. Normally you would use gcc directly to compile you C programs. However the compiling process to make a .gba file is a little complex for you to compile by hand. This is why we have provided for you a Makefile that will *compile*, *link*, and run your program on the emulator with one little command “make vba”.

The only thing that you must do is set up our Makefile to build your program. So you must edit the Makefile and change the PROGNAME and OFILES to compile your program. Quick summary of what you must do

PROGNAME should be the name of the generated .gba file. EXAMPLE: HW9.

OFILES should be a list of .o files (space separated) needed to be linked into your gba file. For each .c file you have put it in this list and replace the .c extension with .o. EXAMPLE: main.o lol.o hi.o

To use the Makefile, place it in the same folder as your other game files, open a terminal and navigate to the directory where the Makefile is and then execute the following command:

```
make vba
```

Deliverables

main.c
mylib.c
Makefile (your modified version)
readme.txt with a brief overview of the game and the controls
Your .c and .h files for the images
and any other files that you chose to implement

Or an archive containing ONLY these files and not a folder that contains these files. DO NOT submit .o files as these are the compiled versions of your .c files

Note: make sure your code compiles with the command

```
make vba
```

Make sure to double check that your program compiles, as **you will receive a zero (0)** if your homework does not compile.

Good luck, and have fun!