CS2110 Summer 2015
Homework 08

Author: **Clayton Pierce**

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.

The content for this homework may not be too complex, but this assignment is a lot of work! **It takes several hours to write a game** that meets the expected requirements, even for a seasoned TA to do so. Do not wait until the day before this is due to start, or you will find that you don't have enough time to design, implement, and bugtest your game by the deadline. It simply can't be done in an afternoon.

Objective:

The goal of this assignment is to make a C program, a game. Your game should include everything in the requirements and be written neatly and efficiently! You will implement the main game loop in **main.c**, use a **mylib.c** file for your functions, but with declarations split into a **mylib.h**, and we encourage you to add as many functions are needed to the file. It is also optional for you to use other .c/.h files to organize your game logic if you wish, **just make sure you include them in submission and Makefile**. 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 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. However, you may use the mylib.h file provided on T-Square if you would like a starting template for your own header.

General Requirements:

1. Unless you know what you're doing, you should implement your game in mode 3. If you use any other modes (4 for double buffering, 0 for tiles, 1 or 2 for affine backgrounds) you *will* be asked questions about how they work during the demo!
2. Images – You must use at least 2 separate images in your game, drawn with **drawImage3**. The images' dimensions must be strictly smaller than 240x160. To do this you must also implement **drawImage3**, which you already created in your previous homework and may use that implementation as long as it works.
3. Your game must contain a fullscreen (240x160) image for the title screen, and a fullscreen image for a win screen/game over screen, whichever applies.
4. You must have **2-dimensional movement** of at least one entity (this entity must be represented by an image from requirement #2). By **2-dimensional movement**, we do not mean one object moves vertically and another object moves horizontally. This single object should be able to move along the x and y axis.
5. You must be able to reset the game to the title screen *at any time* using the select key. You will not get credit for this requirement if there is any time during the game (win screen, game over screen, during gameplay) where you can't return to the title screen by pressing select.
6. You must create a header (**mylib.h**), and put any #defines, function prototypes, typedefs, and extern statements in this file. Remember that function and variable definitions should not go in header files, just prototypes and extern variable declarations. You must write those functions and declarations in **mylib.c**.
7. You must use at least one **struct** to represent an entity. The DMAREC struct from Bill's lecture code will not count toward this requirement.
8. **Button input** should visibly and clearly affect the flow of the game.

9. You should implement some form of object collision, and it should be visibly accurate, given the sizes of the entities colliding. For instance, just checking obj1.x = obj2.x & obj1.y = obj2.y is not sufficient to accurately detect 2D collisions because they can then overlap to a degree before a collision takes place.
10. You must implement the **waitForVBlank** function using the scanlinecounter macro.
11. Use text to show progression in your game, which updates in real-time. Use the example files from lecture, or you can look into text systems more on Tonc: http://www.coranac.com/tonc/text/text.htm
12. **There must be no tearing in your game.** Make your code as efficient as possible! Although these aren't direct requirements for this homework, here are a couple of ways to make your game faster to prevent tearing:
    - Sync your game using waitForVBlank. It isn't even possible to implement the game without tearing if you don't vsync it using a function like waitForVBlank.
    - Reimplement your drawImage3 function with DMA instead of for loops to increase their speed by about ~4x. If you use DMA, be prepared to explain how it works during demo.
    - Only redraw what needs to change on the screen. The vblank period isn't nearly long enough to redraw the whole screen before the next vdraw, even if you use DMA.
    - If you're really ambitious, look into using mode 0. This would tremendously speed up any game, but is significantly more difficult to implement and maintain, and won't be covered in lecture. You'll have to read about it on TONC if you want to use it.
13. Include a **readme.txt** file with your submission that briefly explains the game and the controls.
14. If any of these requirements don't apply to your chosen game, then you must either change what game you are making or change the game itself so that it implements the requirements.

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 your 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 make slight modifications to lecture code and call it your game. Games that are 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 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 you have an idea you would like to check, please consult with the TAs first.

Images

As a requirement you must use at least 2 images in your game and you must draw them all using drawImage3. Your bmptoc program should be able to generate the files necessary to use the images, but in the case that your submission didn't work or in case you would like the convenience of not having to first convert images to bmptoc's specific BMP format, we have provided a tool that will make this task easier for you.

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

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 on next page)

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.

drawImage3

In your game you must use drawImage3.

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 :)
}
```

Protip: If your implementation of this function does not use all of the parameters that are passed in then you're probably doing it wrong.

C coding conventions:

1. Do not jam all of your code into one function (i.e. the main function)
2. Split your code into multiple files (have all of your game logic in your main file, library functions in mylib.c with declarations in mylib.h, game specific functions in game.c)
3. Comment your code, comment what each function does. The better your code is the better your grade!

Warnings

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, covered in Advanced Lab 2.
2. Only call waitForVBlank once per iteration of your while/game loop
3. 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 game is showing signs of tearing!

I **STRONGLY ADVISE** that you go **ABOVE AND BEYOND** on this homework. **PLEASE do not just clutter your screen with squares that run around with no meaning in life** – such submissions will **lose** points. Make some catchy animations, or cut scenes! These games are

always fun to show off after the class, you can even download emulators on your phone to play them. Also, **remember 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, however you may not copy large swaths of code, only use it as a reference. The author assumes you are using his gba libraries, which you are not and may not.

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
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 and furthermore 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

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.

Deliverables

main.c
mylib.c
mylib.h
Makefile (your modified version)
readme.txt
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, you can type ‘make clean’ to remove those 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, and it must compile using the standard CS 2110 flags listed in the syllabus:

-std=c99 -Wall -pedantic -Wextra -Werror -O2

As long as you only change the OFILES/HFILES lines in the provided Makefile, then you should be fine with those above flags. Good luck, and have fun!