

# Homework 10

**This assignment is due by:**

Day: Tuesday November 4, 2014

Time: 11:54:59pm

## Rules and Regulations

### Academic Misconduct

Academic misconduct is taken very seriously in this class. Homework assignments are collaborative. However, each of these assignments should be coded by you and only you. This means you may not copy code from your peers, someone who has already taken this course, or from the Internet. You may work with others **who are enrolled in the course**, but each student should be turning in their own version of the assignment. Be very careful when supplying your work to a classmate that promises just to look at it. If he/she turns it in as his own you will both be charged.

We will be using automated code analysis and comparison tools to enforce these rules. **If you are caught you will receive a zero and will be reported to Dean of Students.**

### 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. Make sure that you submit and demo your assignment by the last demo date of the class. We will not demo anyone past that date, and all unfinished homework assignments will receive a score of zero.

### General Rules

1. In addition 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.
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 in 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

## Starting the homework:

We do not want you to build on your last homework, but make sure you keep the same set up with your files as you did before in Homework 9. That means keeping a similar set up for your VideoBuffer, macros, REG\_DISPCNT, typedefs, and use of a main.c and myLib.c, and also now a myLib.h file. You should **add DMA functions** to these files. Your main.c file should be something different, since in this homework you will be creating your **NEW AND DIFFERENT** own game, but keep the core setup. You can still use the same myLib.c file as you did before, but with declarations split into a **myLib.h**, and we encourage you to add even more functions 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 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: | . |

**Homework 10 is a new game using DMA. You may NOT simply improve your old game.**

**Highlights: DMA, .h files, and structs, oh my!**

## Requirements:

1. Images – You must use 3 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.
  - d. All images (that is the image file that you have ran CS2110ImageTools.jar on, the .png/.bmp/.jpg etc file) should be included in your submission as well.
2. You must also implement **drawImage3** with **DMA**. The prototype and explanation is explained later in the assignment
3. You will maintain mostly the same project structure as for your previous homeworks, but will create a new **myLib.h** file that will contain the declarations for things defined in myLib.c. You will then include this file where needed instead of copying the declarations themselves into each file. Feel free, and it is good practice, to put related code into their own .c/.h files. Remember that function and variable definitions should not go in header files, just prototypes and extern variable declarations.
4. You should use **Structs** to display at least three different objects on the screen at once.
5. Button input should visibly and clearly affect the flow of the game.
6. You still need to have moving objects, etc.
7. You should implement some form of object collision, usually to help progress your game. There are many ways you can make some sort of collision detection, feel free to ask the TAs for help with it.
8. 2-Dimensional movement of at least one entity, be it a player's avatar or an enemy.

9. Try to avoid tearing of the image – implement a function that waits for vBlank, as was demonstrated in lecture, to make the game run smoother. DMA will also help.
10. Use text to show progression in your game. Use the example files that were provided for you during lecture, and you can look into it more in Tonc:
  - a. Check Tonc for more help on this: <http://www.coranac.com/tonc/text/text.htm>
  - b. You may also look at the lecture slides
11. Include a readme.txt file with your submission that briefly explains the game and the controls.
12. Creativity and sophistication – the more time you put into this, the better the grade.

## 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 – Jar available on T-Square

CS2110ImageTools.jar – Jar available on T-Square > 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 the command line by changing directory to where you downloaded CS2110ImageTools.jar, then telling Java to run the jar file, like this:

```
cd ~/Downloads          // Or whatever you downloaded it
java -jar CS2110ImageTools.jar
```

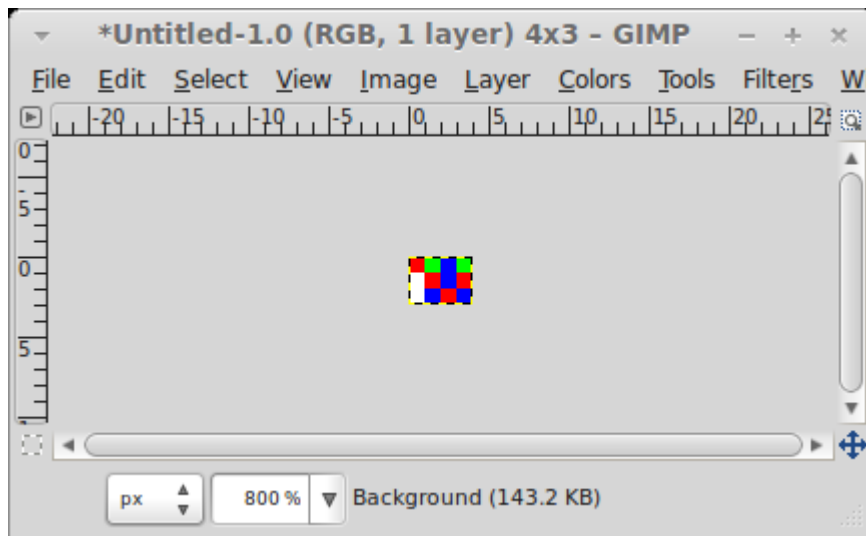
CS2110ImageTools.jar will give you a graphical interface to load image files and save their converted `.c/.h` files.

The output of this program will be a `.c` file and a `.h` file. In your game you will `#include` the header file. In 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 an 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.

## DMA / drawimage3

In your game you must use DMA to code drawImage3.

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 Bill/Tom goes over this in class (Tuesday) 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. However your final answer for drawImage3 must use DMA.

As for restrictions on 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 of the parameters that are passed in then **YOU'RE DOING IT WRONG**.

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.

## 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. If you choose the second option, you will have to create it with primitive shapes since we have not covered drawing images onto the screen. You can have fun with it and build some familiar shapes from the games using pixel art, but we do not expect you to draw a masterpiece with it. It is best to focus on the core requirements of the games. Here are some previous games that you can either create or use as inspiration:

**There are additional requirements to create the game, but most will overlap the given requirements.**

**Doodle Jump:** [http://en.wikipedia.org/wiki/Doodle\\_Jump](http://en.wikipedia.org/wiki/Doodle_Jump)

### Requirements:

1. Accurate, Efficient  $O(1)$  Rectangular Collision Detection implemented as a function.
2. Lives, you can use text to show it.
4. Game ends when all lives are lost. Level ends after some height reached
5. Different types of enemies
6. Structs for platforms, player, enemies
6. Smooth movement

**Dyad:** <https://www.youtube.com/watch?v=hWm-dbYwEpg>

### Requirements:

1. Accurate, Efficient  $O(1)$  Rectangle Collision detection as a function.
2. Smooth motion for enemies and player
3. Latch on to shining nodes to boost forwards!
4. Enemies moving at different speeds.
5. Sensible, repeating patterns of enemy motion, you should have enemies that move in a pattern.
7. Enemies and the Player represented by Structs

**Dig Dug:** [http://en.wikipedia.org/wiki/Dig\\_Dug](http://en.wikipedia.org/wiki/Dig_Dug)

### Requirements:

1. Enemies and the Player represented by Structs
3. Lives, you can use text to show it.
2.  $O(1)$  collision detection with any object. If the player collides with an enemy, it loses a life.
3. Destroy enemies by inflating them
4. 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 enemies are destroyed



**Smash Bros (DO IT!):**

[http://en.wikipedia.org/wiki/Super\\_Smash\\_Bros.](http://en.wikipedia.org/wiki/Super_Smash_Bros.)

**Requirements:**

1. All Nintendo characters. All of them.
2. Multiplayer.

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.

## **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 intensive, 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 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 (available at <http://www.coranac.com/tonc/text/>), 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.

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.

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!

Here the inputs from the GameBoy based on the keyboard for the default emulator vbam (mednafen is different):

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>

## Deliverables

- main.c
- myLib.c
- myLib.h**
- Makefile (your modified version)
- 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.

Good luck, and have fun!