CS2110 Summer 2014
Homework 8

This assignment is due by:

Day: Tuesday July 15, 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.

Overview

The goal of this assignment is for you to develop a small GBA game. You can either pick a game from the list that we provide, or you may develop any other game that you wish, as long as it meets the requirements. You can read more about the actual game requirements of the homework in the "Your Goal" section.

Please note: You are required to create a **new** game for this homework, without using any visual assets from Homework 8. This game has to be **completely different** from the previous assignment. You may still use and build upon your library file that you wrote in the previous homework. You may also inspect the header files and other resources (such as font.c) from lecture, but you **may not** copy significant chunks of lecture code that deals with actual game logic or progression.

Your Goal:

You are going to be making a small GBA game. You may either pick one of the sample game ideas below (**Option A**), or you may make a different game (based on some other game or on an original idea) (**Option B**). **Regardless of which option you pick, your game must still meet the technical Game Requirements** listed below. If you choose to make a game that is not listed below, you **must first email/ask your grading TA** to check if your game will be acceptable. Your game must be similar in complexity to the listed games. If the game does not follow the requirements 100%, but sounds like a really cool idea, then your TA will allow you to make it.

Please Note: While both Option A and Option B are going to be graded by the same criteria, we ***highly*** encourage you to make something unique or different from the game suggestions below. They are simply there in case you are unable to come up with any ideas. If you decide to do Option B, you can make something that you personally really enjoy, potentially making this assignment more fun. This is one of the most open-ended assignments of this course, and so there are many options available to you. Whichever option you choose, you must follow the rules listed below in order to not lose major points. 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.

Game Requirements:

1. You should use **Structs** to display at least three different objects on the screen at once.
2. You should implement some form of **object collision**.
3. **Implement drawImage3 (see below) with DMA**, and use it to draw some elements on the screen, like your player or the enemies.
4. **2-Dimensional movement** of at least one entity, be it a player's avatar or an enemy.
5. **Button input** should visibly and clearly affect the flow of the game.
6. **Reset button (use the "Select" button) to reset the game,** back to the title screen.
7. The game needs to **display the player's remaining Lives**, or some other immediate indication of progress. If the player only has one life, then you may display the score.
8. **Display Title, Win, and Lose screens** converted from images and drawn with DMA.
9. **Use of text** – use the example files that were provided for you during lecture.
10. There needs to be a **sense of progression** – the game should be going somewhere.
11. 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.
12. Include a **readme.txt** file with your submission that briefly explains the game and the controls.
13. **Creativity and sophistication** – the more time you put into this, the better the grade.

Images

You are REQUIRED to put images into your game, meaning that the use of an image package is required for this assignment. There are several different tools that you can use to achieve this. Past semesters used BrandonTools (https://github.com/TricksterGuy/brandontools), which is able to support a variety of formats and modes, but can sometimes be tricky to install. This semester we are encouraging the use of CS2110ImageTools.jar, which can be found on T-Square and is easy to use, but which does not support as many features. Yet another option is to use jGrit, which is both powerful and flexible. Whichever program you choose, you must draw your images with DMA.

If you decide to use BrandonTools (https://github.com/TricksterGuy/brandontools), here is an outline of how it works. Most of this information is still applicable to other applications.

Should be obvious who this is written by. BrandonTools (for lack of better name) handles pretty much any multimedia format (including pdf, avi, ppt) and exports the file into a format the gba can read (in the case of such files you will get an array for each page/frame/slide). Also supports manipulating the images before they are exported.

Download the version that is appropriate for your system and follow the instructions inside the archive.

Before using the program, make sure you read the readme which explains program usage in detail.

Basic usage for mode 3 is:

```
brandontools -mode3 filename imagefilename
```

example:

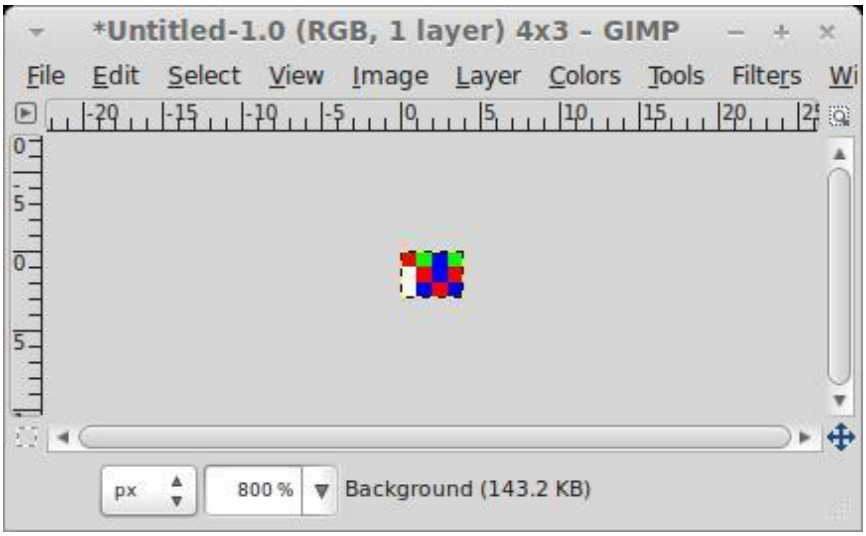
```
brandontools -mode3 myimages kirby.png
```

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 brandontools 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.

For image files I don't care where you get them or you can make them yourself. Here is a link with some image files

http://www.gamedev.net/community/forums/topic.asp?topic_id=272386

http://gamemaking.indiangames.net/index_files/FreeSpritesforGames.htm

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 recieve 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.

Lastly note that we don't care if the background of your image is being drawn. If you want a transparent background for your image then you will need to resort to using sprites. Of course, this is way beyond the Call of Duty for this assignment so if you get this working then more power to you http://www.coranac.com/tonc/text/regobj.htm

Option A: Pick and Implement one of these games:

The World's Hardest Game

The World's Hardest Game (that is actually what it's called) is a simple game where the only task is navigating a small square through a field of moving circles. If you hit a circle, your square starts back in the starting position and a death is recorded. You are free to decide the layout of the levels that you create and the motion of the enemy circles. There is no limit to the amount of times that you can die in this game.

For more information, go to http://www.addictinggames.com/actiongames/theworldshardestgame.jsp.

Play at your own risk. This game is HIGHLY addictive.
Great Additions: Keeping up with deaths, multiple difficulty levels, checkerboard pattern for background of levels (behind moving things), pieces that you have to get before you can win

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.
5. Sensible, repeating patterns of enemy motion **You should have enemies that move in a pattern.**
6. Multiple Levels
7. Enemies and the Player represented by Structs

Implementation hints.
You don't have to follow this, but here are gentle nudges to get you started. There are many paths to a solution; here is one. You should make use of Structs for the enemies. Each enemy should "know" his position on the board and his limits of motion. Don't get caught up trying to make a checkerboard pattern work.

For collision detection, you must implement it as if-statements that compare the positions and dimensions of the two objects. Doing it any other way will reduce performance. Rectangular collision detection between two objects is O(1). You are to implement this as a function as per the requirements above.

CubeField

In CubeField the player tries to stay alive as long as possible in an infinite field of enemy cubes. You start off at a fairly low speed, and get faster and faster as the game progresses. There is no limit to how far left and right you can go, but all forward motion is determined by the game (no speeding up or slowing down manually). When you hit a block you see a game over screen.

For more information see http://en.wikipedia.org/wiki/Cubefield

Great Additions: Make the cubes appear 3D, Change colors as the game progresses, scoring System

Requirements:
1. Must be able to move left and right forever (walls of screen should not restrict you).
2. Accurate, Efficient O(1) Rectangle Collision detection implemented as a function.
3. There must be regular intervals of recognizable patterns, and they should generate in front of you regardless of how far left or right you have gone.
4. You must increase in speed as the game progresses.
5. Smooth motion (shouldn't see any jumping or skipping)
6. Game ends when you hit a cube.

Implementation hints:
You don't have to follow this, but here are some gentle nudges to get you started. There are many paths to a solution; here is one.
Don't think of the board as a constraint. Instead, find a way to keep track of position that is not necessarily dependent on the scene dimensions.
For collision detection you must implement it as if-statements that compare the positions and dimensions of the two objects. Doing it any other way will reduce performance. Rectangular collision detection between two objects is O(1). You are to implement this as a function as per the requirements above

Galaga

The player controls a ship that moves side to side on the bottom of the screen and shoots up at enemies. The enemies move slightly and periodically try to attack the ship by rushing towards it.
Contact with an enemy takes one life from the player. There are multiple levels to the game, and on each level there can be different types of aliens.

For more information see http://en.wikipedia.org/wiki/Galaga

Great additions: Power ups, Bonuses, Scoring, High Score indicator, See ships that you have left.

Requirements:
1. Accurate, Efficient O(1) Rectangular Collision Detection implemented as a function.
2. Lives
3. Multiple levels with different enemy configurations.
4. Game ends when all lives are lost. Level ends when all aliens are gone.
5. Different types of aliens **there should be one type of alien that rushes towards the ship and attacks it**
6. Smooth movement (aliens and player)

Implementation hints:
You don't have to follow this, but here are some gentle nudges to get you started. There are many paths to a solution; here is one.

Use an array to model a level's configurations and structs for aliens. They should know if they are alive, dead, or attacking, for example. Bullets that the ship shoots should also be structs. For collision detection you must implement it as if-statements that compare the positions and dimensions of the two objects. Doing it any other way will reduce performance. Rectangular collision detection between two objects is O(1). You are to implement this as a function as per the requirements above.

Tetris

A random sequence of tetrominoes—shapes composed of four square blocks each—fall down the playing field (a rectangular vertical shaft, called the "well" or "matrix"). The objective of the game is to manipulate these tetrominoes, by moving each one sideways and rotating it by 90 degree units, with the aim of creating a horizontal line of ten blocks without gaps. When such a line is created, it disappears, and any block above the deleted line will fall. When a certain number of lines are cleared, the game enters a new level. As the game progresses, each level causes the tetrominoes to fall faster, and the game ends when the stack of tetrominoes reaches the top of the playing field and no new tetrominoes are able to enter. Some games also end after a finite number of levels or lines.

For more information see http://en.wikipedia.org/wiki/Tetris
Great additions: Ghost pieces, Hard dropping, Holding pieces, Scoring System, Increasing difficulty.

Requirements:
1. Accurate, Efficient O(1) Tilebased Collision Detection. (implement this as a function; see hints).
2. Rotation/Movement. You are able to rotate tetrominoes by 90 degrees and you should be able to move pieces sideways
3. Should be able to see next piece
4. Should be able to clear lines and pieces should fall after clearing lines.
5. Movement should be clear and smooth
6. Game ends when pieces reach the top of the field

Implementation hints
You don't have to follow this, but here are some gentle nudges to get you started. There are many paths to a solution; here is one.
You should have a structure representing a tetromino. It should have a 4x4 array and know its tile coordinates on the matrix.
You will have a 4x4 array (remember the “I” piece is height 4). As an example, here is the box piece:

0 0 0 0
0 1 1 0
0 1 1 0
0 0 0 0

The 0's you don't draw; the 1's you do draw. It’s easier to work
with blocks, or tiles here. As for the matrix: represent that with a 2D array. When a block collides with something, it should stop. When this happens, you just copy the tetromino array into the matrix (remember not to copy any “0”'s). As for rotations, you simply rotate the 4x4 array 90 degrees; however you should check beforehand if the rotation will collide with anything or go outside bounds. If it does, you don't do the rotation.
Every few frames you should move the tetromino down by 1 tile.

Frogger

The player takes the role of a little froggy who is trying to cross a road and a river (or a railroad track and flow of lava) to get to a home on the other side. The challenge comes with a bunch of traffic
on the road that the little froggy can't get hit by and logs and lily pads in the river which are the spots on which the little froggy can land.

Here's a Frogger-style game as an example,
http://www.oldgames.dk/freeflashgames/arcadegames/frogger.php

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.
6) You are not required to use images – boxes and other geometric shapes are sufficient.

Minesweeper

The player is given a grid of squares that represents a minefield. It is unknown to the player which squares contain mines. When the player chooses a square that doesn't have a mine, a number
representing how many mines are adjacent to that square horizontally, vertically, diagonally is displayed on the square. If the player chooses a square with a mine, it's game over. However, if the
only remaining spaces all contain mines, then the player wins.

Here's a minesweeper clone as an example,
http://www.mine-sweeper.com/MineSweeper-Beginner.asp

Requirements:
1. Must have 3 different difficulty levels where the board is populated with more mines on harder difficulties. Harder difficulties will also have bigger boards.
2.When the game is over, you must show the user all of the mines on the field. The game is won if the only uncovered spaces left are mines.
3. The player must be able to mark a square with a flag to indicate they know it has a mine and mark a square as unknown to indicate they don't know what's under that space
4. When the player selects a space that is not a mine it should reveal to them how many mines are adjacent to that space. If there are no mines then it should reveal all of its adjacent neighbors. (see Note).
5. The game must keep track of how long the player has been playing as a way to track his or her score. (Hint; there are roughly 60 vblanks per second).
Note: Although not required but would be great. If there are no adjacent mines then it should also reveal the surrounding spaces recursively. We are only making you reveal that mine's neighbors. In a real minesweeper game any space that is connected to the space that was selected should be revealed.

Option B: Implement Your Own Game

Note: If you pick this option, you **must** ask your grading TA if your game idea is acceptable. But do not be discouraged by this – simply ask us, and we will let you know! We love hearing game ideas and suggestions!

Your game must satisfy the Game Requirements that are listed in the "Goals" section. This game should be of a similar complexity like that of the games in Option A.

Some cool games that people made in the past include Fire Emblem, Skyrim, StepMania, Roguelike, 2048, and various different platformers. Make something cool! It does not have to be a replica of the game. For example, the Skyrim game was a side-scrolling platformer where you could battle a dragon. Just think of something cool and check with a TA if it would make a good game for this assignment!

Warning

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).

I **STRONGLY ADVISE** that you go **ABOVE AND BEYOND** on this homework. **PLEASE do not make Pong** – such submissions will **lose** points. Draw something amazing, or implement something really cool! This is the last GBA homework that you will be working on this semester. Now is your time to show us what you can do and to make us proud! A part of the grading criteria is based on how creative and sophisticated your game is (and how much thought you have put into it), so please try not to rush through this assignment.

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.

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, game specific functions in game.c)
3. Comment your code, comment what each function does. The better your code is the better your grade!

GBA Key Map

In case you are unaware, the GameBoy Advance buttons correspond to the following keys on the keyboard.

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

Additionally, holding the space bar will make the game run faster. This might be useful in testing, however the player should never have to hold down spacebar for the game to run properly and furthermore there is no space bar on the actual gba.

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”.

However 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
```

Alternatively you can execute this command to use wxvbam (another GBA emulator)

```
make wxvba
```

Deliverables

main.c
mylib.c
Makefile (your modified version)
Your .c image files
and any other files that you chose to implement
readme.txt (briefly explain the game and the controls)

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 we are unable to compile your program.

Good luck, and have fun!