Meeting with mbed designer

* Want to learn how to use the library, particularly with regards to the LCD and SD card, and in my case, Bluetooth/UART.
* Have been using DigitalSignage and SprintGame for reference, and would love an explanation on how the main body of the code works i.e. where the libraries are being used.
* If there are specific lines/functions/classes we especially don’t understamd, then can we get a more in-depth explanation of what they do?
* If we don’t understand it, but we know how to use it, is that enough?
* Specific queries to be checked:
  + How is the LCD initialised and then drawn to?
  + How does image synthesis work with SprintGame?
  + How is file conversion working? (file format, conversion tool, but SD?)
  + How does the game timer for SprintGame work? Does it use SystemTimer, and how does that work? Can it be paused?
  + Do you recommend using the event system/creating our own specific structs?
* Queries specific to me:
  + How can a UART connection be established?
  + Are there any libraries for handling gamepad input?
  + Is there a way to see the controller output?
  + Is using the digitalIn button in place of the controller easy to do?
  + Are my ideas for collision detection/jumping/appearance of obstacles correct?

# Main analysis

# Library usage

## LCD

Performing the concept of scrolling involves using both video buffers, where you have one for current display and one for the next frame.

From Swipe Runner (SprintGame):

Made up of the main, the image table and imported libraries.

If you want to move the LCD, just import the 3 libraries in the game. For the LcdCfg file, don’t touch the macros for the LCD parameters, because you don’t need to touch it.

The mbed.h header is the same for os library or standard

Use button fall for a falling edge interrupt so that it understands when the user button has been pressed

There is an interruptIn class for changing pins to be interrupts. The ports which can be used for interrupts are decided by the RZ/A1.

Leave the init as it is.

Next bit is assigning 2 separate buffers and making the count 2.

Canvas2D is an important pointer for writing with RGA

580: use everything up to this point and will have no problems.

578: If the init and backlight are done at the same time, a lot of voltage is used at once, so we wait a slight bit to make sure it doesn’t blow up.

Call-back seems to be the name for interrupts in this instance.

Threads are used for when you want to perform an interrupt in the sense that they are always working so that you don’t have to worry about them.

Disp\_info is a struct which handles the entire drawing.

Drawing is all done by canvas2D. First you clear the whole background, and then globalalpha is how transparent the next picture is, where 1.0f is solid, 0.0f is unseen.

To make my pictures, I could make a whole slide size and then transfer it to gimp and reduce it. Make the background transparent, and then copy paste. Increase the size to fit the image, then cut using the canvas size tool. Afterwards cut and export.

The transparent parts are likely cut by the RGA class, but may take longer. See bottom of SD.

Always call update once everything has been written to update the display. Might not need to use wait\_Vsync. This is used to ensure that the switch between buffers and the writing is all safely finished.

## SD

Don’t use mbed os library, delete it first, and then search for the mbed library, because SD reading is a bit slow. Double click the mbed library to import. Might need to also import mbed-rtos.

Define storage type as user 0 so SD becomes default. The GR-PEACH wrapper library is better because the way of doing it is the same but faster and easier.

The mount name macro is free and you can name it however you want.

61: starts here, you call a new storage object using the storage function

75: keep waiting for the connection

78: you use the mount unmount so that it keeps moving everytime you take out the card

91: searches using a general function called opendir on the above folder path macro

88: fp starts off as null but once it finds a file it will point to it. D is a file pointer

94: checks for a file that ends in .wav and reads if it is. It basically searches the file path until it finds a .wav and runs it

DS uses a scan folder library for audio, which is used to go back instead of just reading in order. You can just enter your file names if you know what they are.

For me, I know the file names, so I set up the storage as above, open the file with the pointer, use fread to read the file and attach it to a variable, then close it, and do the same for all of them. For “No data”, I can use the conversion tool and then define where it is in the binary image file by address (well, 0x0000). Make sure to use the extern C file to ensure it can be read, since it’s not a .cpp file.

Could be best to use the tool and save the binary and the header to the SD, and then pull it from SD so it becomes the same state. Could even create binary similar to the header, basically all you need to know is the offset from the start of the file and move the address locations to RAM. By taking the XML file inside the tool and changing it so that the output format is ARGB8888 then I don’t need to do anything hard. Could even make it ARGB4444 so it’s even lighter.

Need to save the data table in the C file as binary in the RAM. To do so, we create a binary table for convenience, which refers to the binary data table stored in RAM, such as:

uint8\_t  BinaryImageDataTbl[ xxxxxxxx ];

(which has a large enough memory size reserved to save all the data – the xxxx part)

Then, we need to create some pointers which correspond to the define statements in the header file, so that we know where each image begins in the binary table we just made. For example:

#define data\_1 ((const graphics\_image\_t\*)( RGA\_Sample\_BinaryImage+ 0x00000000 ))

#define data\_2 ((const graphics\_image\_t\*)( RGA\_Sample\_BinaryImage+ 0x00025820 ))

would become:

data\_1’s pointer would be &BinaryImageDataTbl[0x00000000] 、

data\_2’s pointer would be &BinaryImageDataTbl[0x00025820]

To reach this conclusion, there are 2 options for the kind of data which we save in the SD card. Either:

1. Change the data table to binary by hand, and save that data to the SD. This makes the program easier but involves more manual labour.
2. Save the C file as is, and read the data out piece by piece while converting to binary, and save the result in the table. This requires less labour, but more complex programming.

Think for yourself about the header file.

The size is written in the BinaryImage file

## UART

To use the serial class, you need to know the size i.e. how many bytes to use the read command, but if you want to read an unknown amount of data, getc could be the best/fastest. Either way, use Serial to create a connection, then call getc and assign the input (a button) to a variable. Use readable to see if there is something to read otherwise it might stop, but it may be better to create a separate task using the Thread class.

In my case, waiting 10ms might be best so it isn’t working the whole time and slowing the CPU down, Upping the priority might also be a good thing. Or wait and poll.

First make the thread somewhere in main, and then the task will only read, but make sure to set up the serial port at the start of the task/before the main (static area).

# Other queries

Adding static to everything to limit the scope to the main is good.

There is a timer class which moves on start. Reset will bring it back to 0. Using read\_ms will show it in millisecond order. There is also a stop function. Made a second timer called stopwatch.

To display the timer itself on the screen, created a table of numbers from 1 to 9 which are image files and a function for creating seconds, then just used draw\_image using the position and the current number in the table. After being done, call the clear error to make sure it listens to the next order if there is an error.

Bituse.info/game for collision, also there is a function for acceleration which can be viewed from hello\_world ease out for jump ease in for fall. Could use ease out the whole time. Since the jump is static, you could just choose how many pixels to move it each time.