HIGH LEVEL PROGRAMMING (ie MODULAR GUI)

1. Single functions
   1. !!!!ALL OF THESE FUNCTIONS REQUIRE FONT SIZE WHICH RANGES FROM 16-31 AND IT GROWS IN SIZE from 16 to ~25 and from ~26 – 31 (ie26 is way smaller than 25)
   2. **startFrame()**

write this function at the beginning of every frame - clears previous frame and handles default values

* 1. **endFrame()**

write this function at the end of every frame - handles end of frame data prottocols to force screen update and flush buffer

* 1. **display\_text(int x, int y, int font, const char\* text)**

displays text at given coordinates

* 1. **display\_textColor(int x, int y, int font, const char\* text, byte R, byte G, byte B)**

displays text at given coordinates with custom color

* 1. **display\_textRight(int x, int y, int font, const char\* text)**

displays text at given coordinates, aligned on the right

* 1. **display\_textColorRight(int x, int y, int font, const char\* text, byte R, byte G, byte B)**

displays text at given coordinates, aligned on the right, with custom color

* 1. **display\_number(int x, int y, int font, int number)**

display number at given coordinates

* 1. **display\_numberColor(int x, int y, int font, int number, byte R, byte G, byte B)**

display number at given coordinates with custom color

* 1. **display\_numberRight(int x, int y, int font, int number)**

display number at given coordinates, aligned on the right

* 1. **display\_numberColorRight(int x, int y, int font, int number, byte R, byte G, byte B)**

display number at given coordinates, aligned on the right, with custom color

* 1. **display\_text\_number(int x, int y, int font, const char\* text, int number)**

displays text aligned on the right (ie staying on the left) and a number just after that

* 1. **display\_text\_numberColor(int x, int y, int font, const char\* text, int number, byte R,** **byte G, byte B)**

displays text aligned on the right (ie staying on the left) and a number just after that with custom color

1. Graph class
   1. **Graph(int x, int y, int width, int height, float dataXMin, float dataXMax, float** **dataYMin, float dataYMax, int dataSize)**

constructor, sets the position and dimensions as well as axis ends, maximum number of points on the graph

* 1. **addPoint(float dataX, float dataY)**

adds a point to the graph at given values

* 1. **overridePoint(float dataX, float dataY, int index)**

overrides a data point at given index, use in the need of data correction

* 1. **display\_graph()**

displays the data, axis and axis ends

* 1. **makeLines()**

transforms the individual points to a connected line strip, be careful about line order as sorting is not implemented

* 1. **makeTimeGraph(float currentTime, float timeWidth)**

transform the graph in a time graph, which can be updated (ie the X axis moved) to show dependence on time

* 1. **updateTime(float currentTime)**

updates the axis ends, translates all of the points by the correct amount and deletes old points

1. MessageLog class
   1. **MessageLog(int x, int y, int font, int numberOfLines, int widthChar, int lineSpacing)**

constructor – sets the char array, position and dimensions

* 1. **newLine(const char\* text)**

ads the text to a new line

* 1. **addText(const char\* text)**

аdd the text to the same line

* 1. **addNumber(float num)**

add the number to the same line

* 1. **addNumberPrecision(float num, int precision)**

add the number to the same line, given a specific precision

* 1. **display\_messageLog()**

displays all of the rows with the bottom one being the most recent one, the second from bottom to top, the second most recent one and so on

LOW LEVEL PROGRAMMING

Our display is run by FT810 Embedded Virtual Engine. It handles all the data transmission to the display module via 24-bit parallel interface (this is ideally as this iteration of the chip only handles 18-bit, but this only reduces the number of colours we can achieve – not a big problem). It is a SPI slave accepting high level graphical commands from the MCU.

We use an open source, free to use library that basically has all of the registers extracted from the datasheet and implemented as constants to program the driver. the library also handles the data transmission of slightly more complicated functions to the driver such as display of text. During the first implementation on a new MCU see the comment at the end. Most useful commands:

1. FT8\_init() - initialises the display and the driver
2. FT8\_cmd\_dl(…) – send a command to the driver
3. CMD\_DLSTART – start the display list – INCLUDE AT THE START OF EACH FRAME
4. DL\_CLEAR – clears the screen – usually this is done at the beginning of every frame
5. DL\_COLOR\_RGB – set the colour for drawing ( use bitwise OR with the colour to apply it to the command – ex DL\_COLOR\_RGB | 0xffffffUL - this is white, there are some predefined colours for easier use)
6. FT8\_cmd\_number( X , Y , Font , 0, number) – Font varies from 16 to 25 and from 26 to 31 (both of these star very small and enlarge)
7. FT8\_cmd\_text( X , Y , Font, 0, "Place static char array here") – Font varies from 16 to 25 and from 26 to 31 (both of these star very small and enlarge)
8. DL\_BEGIN – starts a drawing vertexes by given mode – use bitwise OR with drawing mode such as FT8\_POINTS, FT8\_LINES, FT8\_LINE\_STRIP, FT8\_EDGE\_STRIP\_R (line strip with all pixels to the edge of the screen filled – available in \_R, \_L, \_A, \_B) and FT8\_RECTS (defined by two edges)
9. VERTEX2F( X \* 16 , Y \* 16 ) – define the vertexes to be drawn (note that all coordinates given have to be 16 times bigger due to an integrated precision factor)
10. DL\_END – ends drawing vertexes

Note – 8,9,10 follow the sequence – begin, vertex….vertex, end

1. DL\_DISPLAY – instructs the driver to run the commands INCLUDE AT THE END OF EACH FRAME
2. CMD\_SWAP – makes the current list of commands active INCLUDE AT THE END OF EACH FRAME
3. FT8\_cmd\_execute() – executes the commands INCLUDE AT THE END OF EACH FRAME

IMORTANT WHEN USING ON NEW MCU

– as of now the code is modified in order to run on Arduino. The only thing that has to be changed within the library is FT8\_confing.cpp where the most fundamental functions of the library such as the SPI data transfer and slave selection trough driving FT8\_CS low and high so MODIFY accordingly to the new MCU commands. The main sketch is also written for Arduino but changing that should be as easy as swapping to the respective commands for the new MCU.

For further questions contact Dimitar Ruzhev at dimitar.ruzhev@sjc.ox.ac.uk