Testing

I have chosen to use two main strategies to test my teletext rendering program: automatic testing and rigorous testing files.

Automatic Testing

Automatic testing is the process of writing testing code into the program as the program is built. This mainly includes writing code into each function to ensure that the function performs correctly.

The purpose of automatic testing is to ensure that the code correctly implements the program design. The advantage of this it that if every function has been auto tested then the programmer can be sure the code mirrors the program design. However, one of the disadvantages of this method of testing is that if the design being implemented is flawed, the program still may not perform to a client’s specifications, even if implemented perfectly.

My automatic testing consists of assert statements inserted at strategic places in the teletext functions. For teletext to work, it is imperative that the control codes set the correct mode configurations before rendering. Therefore, the bulk of my assert statements ensure that the functions dealing with control codes set the correct mode configurations. The subdirectory teletext/testing contains the testteletext.c file that contains the assert statements and a makefile to run it.

Testing Files

My second testing strategy is to use rigorous testing files to ensure that the teletext control codes render according to the provided specifications. These are simply .m7 files that contain teletext hex codes in various reasonable combinations to ensure that they render correctly for a user.

The advantage of this strategy is that it will discover flaws in the design of the program itself. If the codes do not render according to the provided specification, and appropriate automatic testing has been used, then the source of the problem is likely to be the logic of the program itself. If all reasonable combinations do render properly, then one can be reasonably sure that the program has been tested to a point where it will perform as correctly in the hands of the user.

I have written separate testing files to test alphanumeric text, contiguous graphics, separated graphics, held graphics, and double height text (these are the .m7 files contained in teletext/testing).

alphanumtest.m7

This file checks that alphanumerical characters print correctly with various colour and background colour changes and that the colour values correctly reset to default on a new line. The file verifies that they render correctly.

During the course of this testing I discovered that *The New Background* control code was setting the foreground colour to the same value as the background colour when an alphanumeric colour code was not explicitly issued. It’s reasonable to assume that the foreground colour would remain white in this instance, I altered the program logic to reflect this, solving the issue.

blockgraphicstest.m7

This file tests that contiguous block graphics render correctly. The file test various combinations of graphics related control codes, including colours, changing colour in graphics mode, changing colour in blast through text and background colours. They function correctly.

During this testing I found that graphics control codes were not printing in the current background colour. In the teletext program I was moving the position along the x axis for control codes but actually printing a space with SDL. I have now rectified this problem.

sepgraphicstest.m7

Tests separated colours. I discovered two problems during this testing.

First, background colours does not display perfectly for separated graphics. In separated graphics the filled area of each foreground cell of the sixel is smaller, leaving a pixel wide border around the cell not filled in either the background or background colour. This is not resolved yet but can be fixed by adding a few lines of code to the fill\_block function in teletext.c.

Second, due to my chosen implementation separated graphics resets to contiguous graphics when a new graphics colour code is issued. If the logic in decode\_block\_graphics is changed to fix this, then held graphics cannot be implemented correctly. Consequently, I have chosen an implantation whereby held graphics has full functionality but separated graphics is reset to contiguous when a new colour code is issued.

holdgraphicstest.m7

Tests the functionality of hold graphics mode in contiguous graphics, separated graphics and blast through text. Tests colour changes, background colour changes and resetting on a new line. Renders correctly.

doubleheighttest.m7

Tests the functionality of double height text mode including colour, background colour and resetting on a new line. I discovered a problem not yet fixed. If there are 4 lines of double height text codes (which should render as two lines of text in double height), the top two render properly but the bottom two only render as the bottom half of the text. This is because my program, on receiving a double height code check the line above. If the line above is also double height, then the current line becomes the bottom half. If there are four lines of double height, then the third line, which should render as the top half, finds a double height code above and incorrectly renders as the bottom half. This could be solved by looking two or three lines above, but the problem would re-occur is there were six lines of double height.