Displaying Apple II Double Lo-Res Bitmapped Graphics in a cc65 C Program

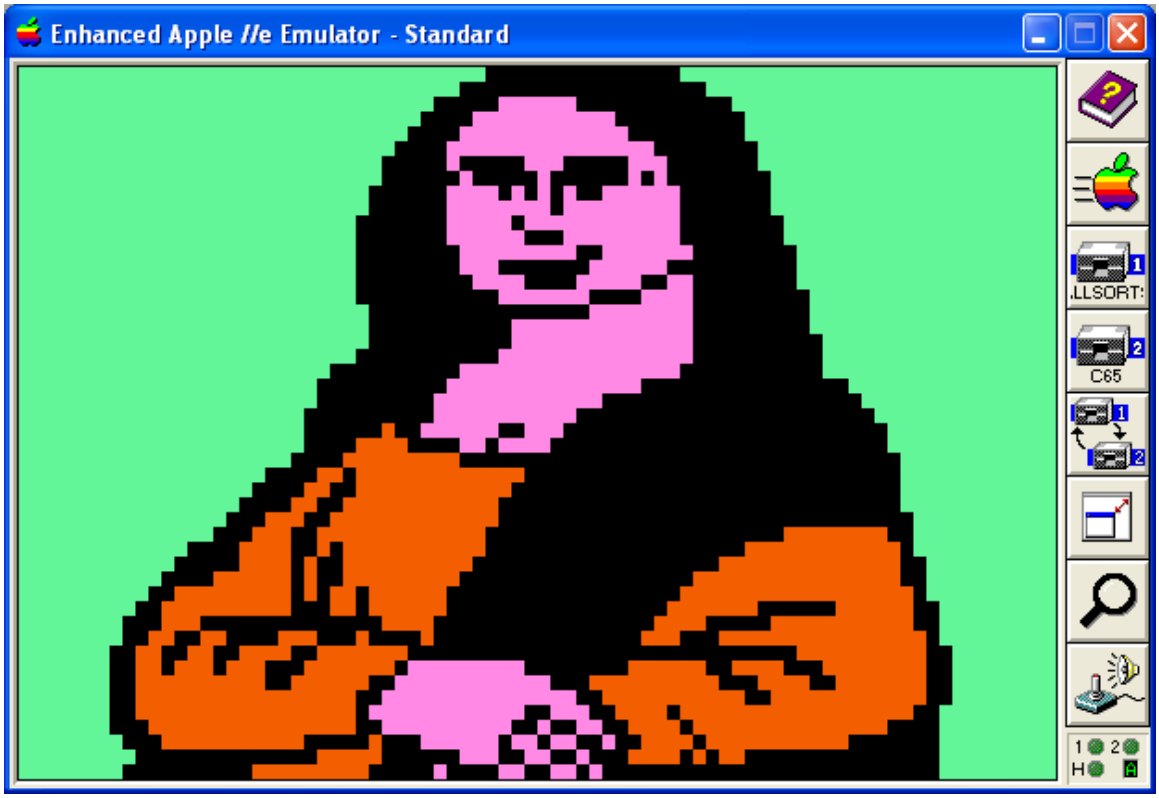
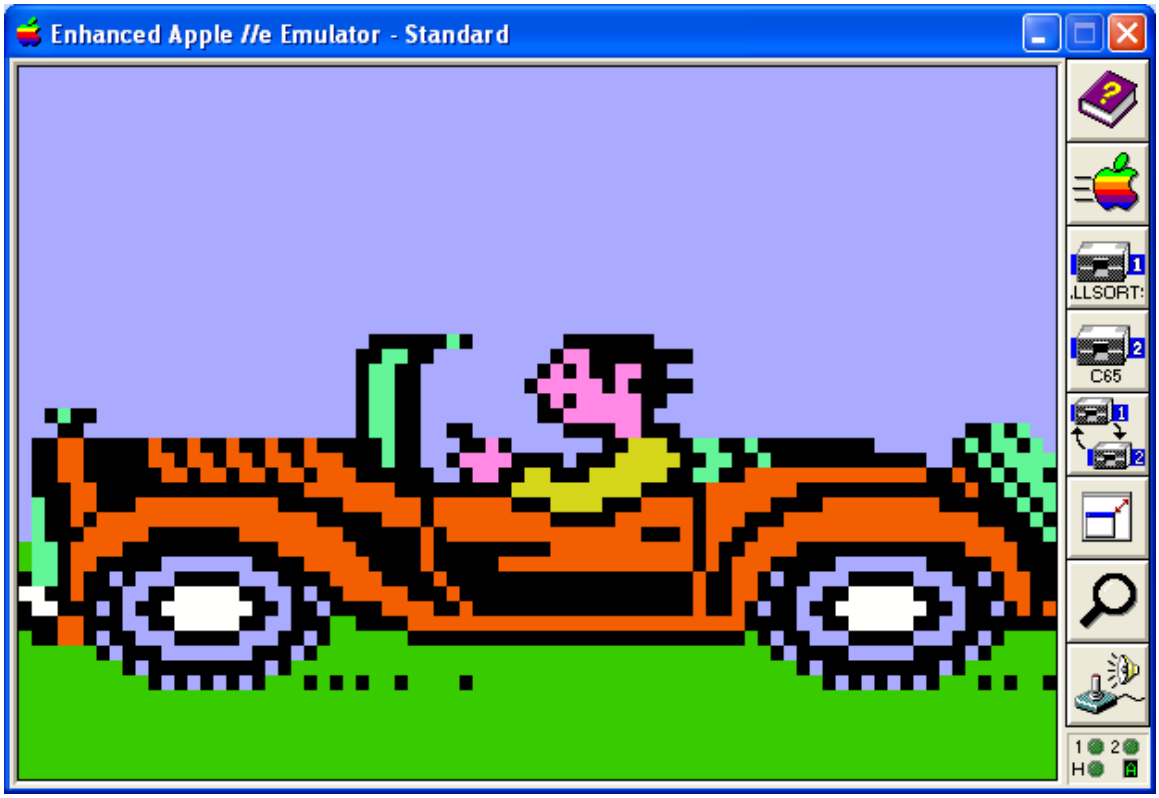


Table of Contents

Forward



Back in 2009, as part of a larger collection of programming for the Apple II, I wrote an MS-DOS utility called BMP2LO.EXE. In a nutshell, all BMP2LO does is remap the colors from a PC bitmapped graphic image to an Apple II graphic image in either Lo-Res (LGR) or Double Lo-Res (DLGR) format and save the results.

BMP2LO outputs a BSaved Image File pair which can be easily BLoaded in an AppleSoft BASIC program, and also outputs a raster oriented image which is more elegant, and more suitable for a C language program outside BASIC's bog and mire.

Double Lo-res Graphics Mode for the Apple II is a pretty decent venue for displaying full-screen colored versions of Minipix (and other “little pictures”). Since Double Lo-res is 80 pixels x 48 rasters, and Minipix are 88 pixels x 52 rasters, this is close enough in size for the BMP2LO.EXE conversion utility to output these so they can be displayed on the Apple II in glorious full-screen detail and 16 discrete colors (after you color them). And since this utility works on Windows 16 color BMP files it is also easy enough to create original Double Lo-Res images of ones own design to be converted.

Last Year (2013) I drastically revised BMP2LO and added several additional utilities to provide round-trip conversion (and editing) of LGR and DLGR files.

These utilities became part of my Apple II Aztec C65 AppleX distribution's 2013 major release along with many demos including DLGR demos for the Apple IIe written in Aztec C65, as well as for my DOS 3.3 equivalent of AppleX which is called Apple33.

More about some of these utilities can be found at the following link:

http://www.appleoldies.ca/graphics/index.htm

The Aztec C65 cross-compilers and most of the utilities I have written for them run in MS-DOS. Over the past year or so as I became an expert on the Apple II I realized more and more the failings of Aztec C65 programs and Aztec C65 itself when compared to the cc65 cross-compiler. I also realized more and more that MS-DOS is as dead as the Apple II, and even emulators like DOSBox don't always work with the oldest Aztec C65 compilers, like the Commodore 64 compiler that I put together and distribute from the Aztec C Website:

http://www.aztecmuseum.ca

In 2009 I also decided to port all of my Aztec C65 work to the cc65 cross-compiler, but I wasn't quite ready. But by May 2014 I had decided to do my utilities exclusively in the MinGW gcc compiler where possible, and abandon MS-DOS where possible. It was time to port everything else related to programming the Commodore 64 and Apple II to cc65.

MS-DOS support is quickly vanishing from the planet. Cc65 writes generally faster and smaller code than Aztec C65, so this move comes none too soon.

This document provides an accounting of the porting of one of my demos from this body of work to cc65; a DLGR Slide-Show demo from last year's AppleX major release.

***

When The Print Shop first appeared on The Apple][, the only type of graphics it used were small 4 sector DOS3.3 files, called minipix. Because few were distributed with The Print Shop®, people drew their own, using the print Shop Graphic Editor, and soon these little 88 x 52 pictures were everywhere. There were disks and disks, just packed with them. When "The New Print Shop" came along, you could convert these little graphics to the new format which was the same size. Over the years I have managed to accumulate a monstrous collection of Minipix which I distribute with my Clipshop program for Windows Users at http://www.clipshop.ca and which I hope you will take time to download and take full advantage of, if only for the Minipix alone.

As noted earlier, since Double Lo-res is 80 pixels x 48 rasters, and Minipix are 88 pixels x 52 rasters, these convert and display well on the Apple II in DLGR. The images used in this demo are converted from colored MiniPix prepared in Windows Paint.

Introduction - DLOSHOW in cc65

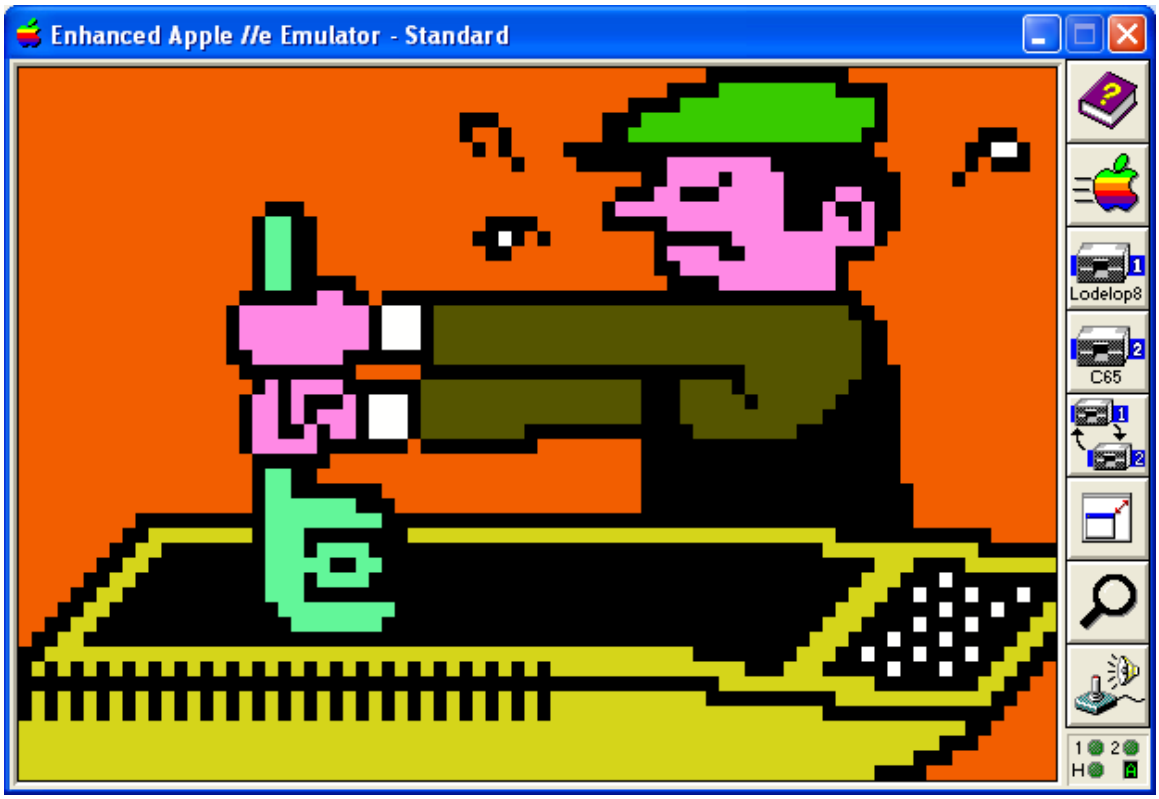
The cc65 Apple II runtime doesn't take kindly to those who step-outside the framework of its internal design. So mucking with the Apple II hardware directly is a careful struggle to hide from cc65 and hope you don't get caught or you are out of business! This is a little different from Aztec C65 which did not concern itself with hiding in every available piece of memory on the Apple II. Neither the cc65 Apple II library, nor the Aztec C65 "stock" library offer support for many Apple II graphics modes.

DLOSHOW Program Organization

When DLOSHOW starts it checks the current directory for a slide-show script called "PICLIST" which is an Apple II format sequential text file with a DLGR file name on each line. If PICLIST is not found, it is automatically created from all the available DLGR files in the current directory. A PICLIST can be created or revised and edited in any text editor that works with Apple II text.

DLOWSHOW then goes on to display the DLGR slide-show. The Slides display continually. When the end of PICLIST is reached it starts over at the beginning. The ESC key can be pressed at any time to exit DLOSHOW and any other key will just advance to the next slide. By default, if no key is pressed DLOSHOW displays a slide for 5 seconds using a timing loop based on a 1 MHZ Apple II, just like a Beagle Bros. slide-show.

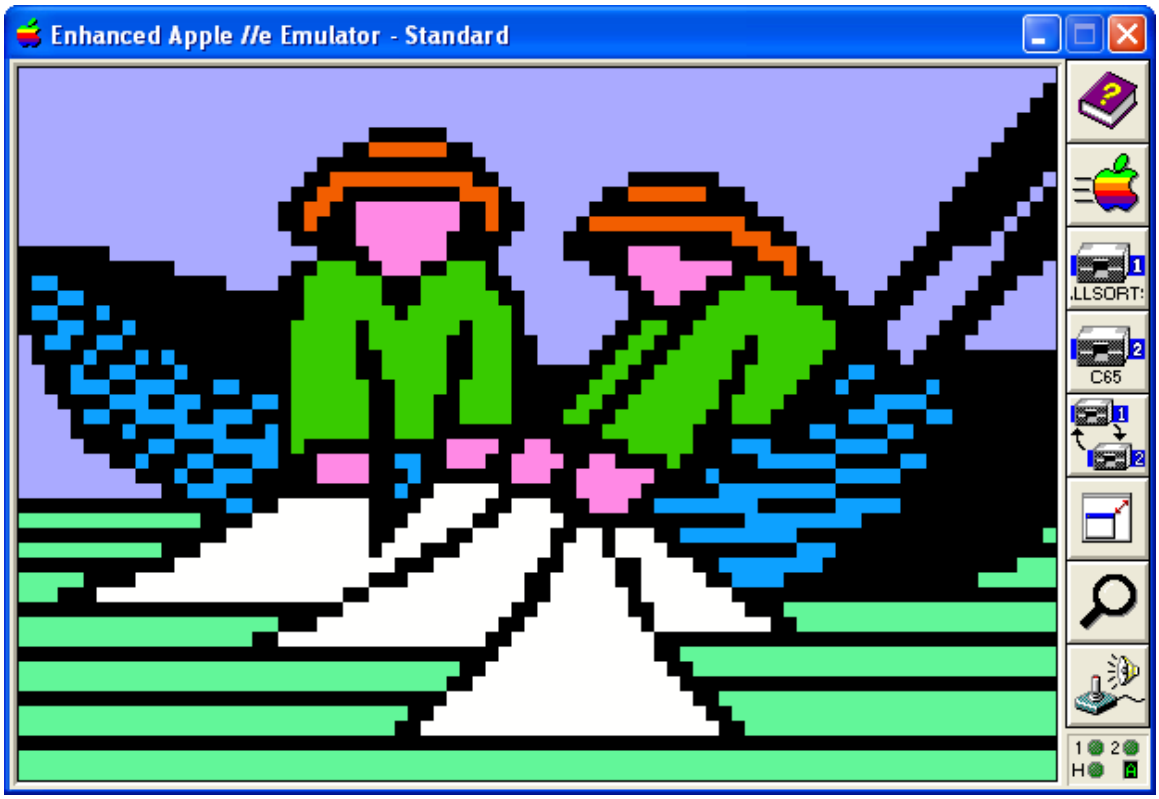
Timing isn't Everything



Actually timing isn't anything. On the Apple II you can have a clock add-on like the no-slot clock but unlike the Commodore 64's TOD (Time of Day Clock which is built-in) we have no such beast that is consistent across Apple II's. So I just did some timing tests and programmed a timing loop for this demo like the Beagles before me☺

```
/* 1MHZ timing loop settings - increase for faster machines */
#define YTIME 60L
#define XTIME 32L
char wait(unsigned duration)
{
    char c = 0;
    long y, x;
    while (duration > 0) {
        for (y=0;y<YTIME;y++) {
            for (x=0;x<XTIME;x++) {
                if (kbhit()) {
                    c = cgetc(); /* clear keyboard buffer */
                    while (kbhit())cgetc(); return c;
                }
            }
        }
        duration--;
    }
    return c;
}
```

Building the PICLIST Slide-Show Script




```
/* the following function creates a list of the double lo-res
   files in the current directory, in Apple sequential text format. */
FILE *makepiclist()
{
    FILE *fp = NULL;
    int err, cnt=0, idx, jdx;
    DIR *dir;
    struct dirent *dp;
    char ch;

    extern unsigned char _filetype;
    extern unsigned _auxtype;

    /* read files in current directory */
    if ((dir = opendir (".")) == NULL) return NULL;

    /* remove piclist if it already exists */
    if((fp=fopen("PICLIST","r"))!=NULL) {
        fclose(fp);
        remove("PICLIST");
    }
```

To set the ProDOS FileType and Auxiliary Type we need to go outside the C programming language's normal file operations and into the ProDOS MLI (Machine Language Interface). The cc65 MLI support is not exposed to the user and at this time no

reasonable way to expose this exists. I will address this in the future, but for now the Create Block in the cc65 MLI layer exports data that can be used to set the FileType and Auxiliary Type before calling ProDOS to create a file:

```
/* set file to sequential text */
_filetype = 0x04;
_auxtype = 0x0000;

if((fp=fopen("PICLIST","w"))==NULL) {
    closedir(dir);
    return NULL;
}
```

As we read through the ProDOS directory file, cc65 provides us with extended information about files in a directory using an extended POSIX-like File Entry in the form of the dirent structure. We are looking for Binary Files (ProDOS FileType $06) with a Load Address (Auxiliary Type) of $0400 which is the address of Page 1 of the Apple II text screen. We are reasonably careful about which files we put into the PICLIST. We only accept two file extensions; DLO and DL1. BSaved DL1 and DL2 files are in pairs, so we only put the DL1 file into the PICLIST and we "cheat" because we don't check for a matching DL2. We also want to verify the file size which is also included in cc65's dirent structure:

```
/* write the list to disk */
while ((dp = readdir (dir)) != NULL) {
     /* binary files only - load address 0400 */
     if (dp->d_type != 0x06 || dp->d_auxtype != 0x0400) continue;
     /* get extension */
     jdx = 999;
     for (idx = 0; dp->d_name[idx] != 0; idx++) {
         if (dp->d_name[idx] == '.') {
             jdx = idx+1; break;
         }
     }
     /* extension must be .DLO or .DL1 */
     if (jdx == 999) continue;
     ch = toupper(dp->d_name[jdx]);
     if (ch != 'D') continue;
     ch = toupper(dp->d_name[jdx+1]);
     if (ch != 'L') continue;
     ch = toupper(dp->d_name[jdx+2]);
     if (ch != 'O' && ch != '1') continue;
     /* filesize must match */
     if (ch == '1' && dp->d_size != 1016) continue;
     if (ch == 'O' && dp->d_size != 1922) continue;
     /* if out of disk space quit writing to piclist */
     err = fprintf(fp,"%s%c",dp->d_name, (char)13);
     if (err < 0) break;
     cnt++;
}
```

Note above that we are creating an Apple II native sequential text file with carriage returns and not a unix text file. The Apple II never used Unix text natively.

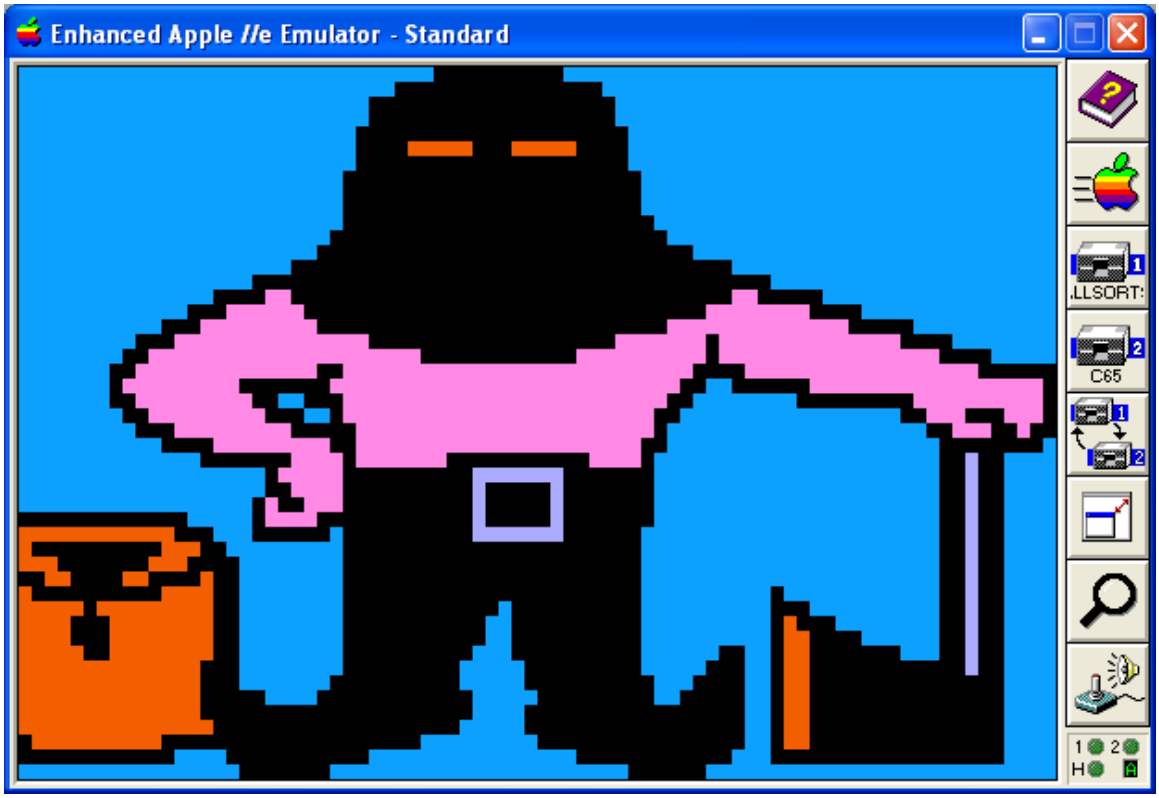
Now that we are done reading the current directory and we have built the PICLIST file we close the file and the directory. If no DLGR files were found, we return a NULL file pointer. Otherwise we return a file pointer to an open PICLIST to start the slide-show:

```
    fclose(fp);
    closedir(dir);

    if (cnt == 0) {
/* if no files found or no filenames written remove empty piclist */
        remove("PICLIST");
        fp = NULL;
    }
    else {
/* otherwise open and return the piclist */
        fp = fopen("PICLIST","r");
    }

    return fp;
}
```

Reading the PICLIST for Apple II Compatibility



I mentioned above that cc65 does not use Apple II native text. Neither does Aztec C65 unless you force it to. Aztec C is a little more robust but still needs coaxing. To avoid any confusion at all we need to correct cc65's behaviour by supplying our own text file reading function; we need to replace the library call fgets() with a local fgets() so the cc65 linker uses ours and not theirs for reading PICLIST when we display the slide-show.

Our focus for writing Apple II Applications whether in C or in AppleSoft BASIC doesn't know or care about Unix text or MS-DOS text. MS-DOS applications use MS-DOS text, and Unix applications use Unix text. So why would an application that runs on the Apple use Unix text? For my DOS 3.3 applications written in C, I use DOS 3.3 text and for ProDOS 8 and beyond on the Apple II, I use ProDOS 8 native text.

The requirement is clear and compilers like Aztec C65 and cc65 are badly behaved guests at times and always need to be beaten into submission to make sure they don't break Apple II system file integrity. Keep in mind also that since I have modified Uz's fgets() right from the cc65 library to repair this deficiency, the cc65 code is no larger or slower than it would have been if we'd settled for POSIX world domination of our beloved Apple II's filing system and just used the fgets() that comes with cc65☺

```
/* fgets for Apple II text files - read native apple II text files
properly! The cc65 version doesn't! Which means that you need to use
unix text files unless you roll your own fgets... */
char* __fastcall__ fgets (char* s, unsigned size, register FILE* f)
{
    register char* p = s;
    unsigned i;
    int c;
    if (size == 0) { /* Invalid size */
        return (char*) _seterrno (EINVAL);
    }
    /* Read input */
    i = 0;
    while (--size) {
        /* Get next character */
        if ((c = fgetc (f)) == EOF) { /* Error or EOF */
            if ((f->f_flags & _FERROR) != 0 || i == 0) {
                /* ERROR or EOF on first char */
                *p = '\0';
                return 0;
            } else {
                /* EOF with data already read */
                break;
            }
        }
        /* One char more */
        *p = c;
        ++p;
        ++i;
        /* Stop at end of Apple II Text File line */
        if ((char)c == (char)13) {
            break;
        }
    }
    /* Terminate the string */
    *p = '\0'; /* Done */
    return s;
}
```

Homebrew – Setting Double Lo-Res

Most Apple II Programmers already know how to peek and poke their way around a little and those still left alive today likely know some assembly language. This stuff is all really well documented. C compilers like Aztec C65 depended heavily on Assembly Language for user extensions and didn't build-in a lot of restrictions behind the scenes because that got in our way. But they got old and slow and fat when cc65 showed-up as the fast-talking "new kid on the block" with "silver-bullets flying all over" and doing really cool things behind the scenes to make everything easier and C-like. The even better news is that if you are careful, you can combine the old with the new, but if you enjoy your HomeBrew as much as I do, you need to hide from cc65 while cc65 hides from you.

You saw the Hack above with fgets(). Now we need to examine another type of workaround that has to do with accessing the Language Card area since cc65 uses that memory for behind the scenes memory management. If you don't use cc65's own call when you switch to 80 column mode to initialize Double-Res Graphics you'll wipe-out cc65 and crash into the monitor.

So instead of rolling our own, we work with cc65 so it is none the wiser when we use the backdoor hardware to display Double-Res mode. Instead of using "PR#3" like we used to do in AppleSoft BASIC to initialize the language card, we use cc65 to set to 80 column mode by calling cc65's setvideomode() function and then we don't worry about cc65 again for the rest of the DLOSHOW program…

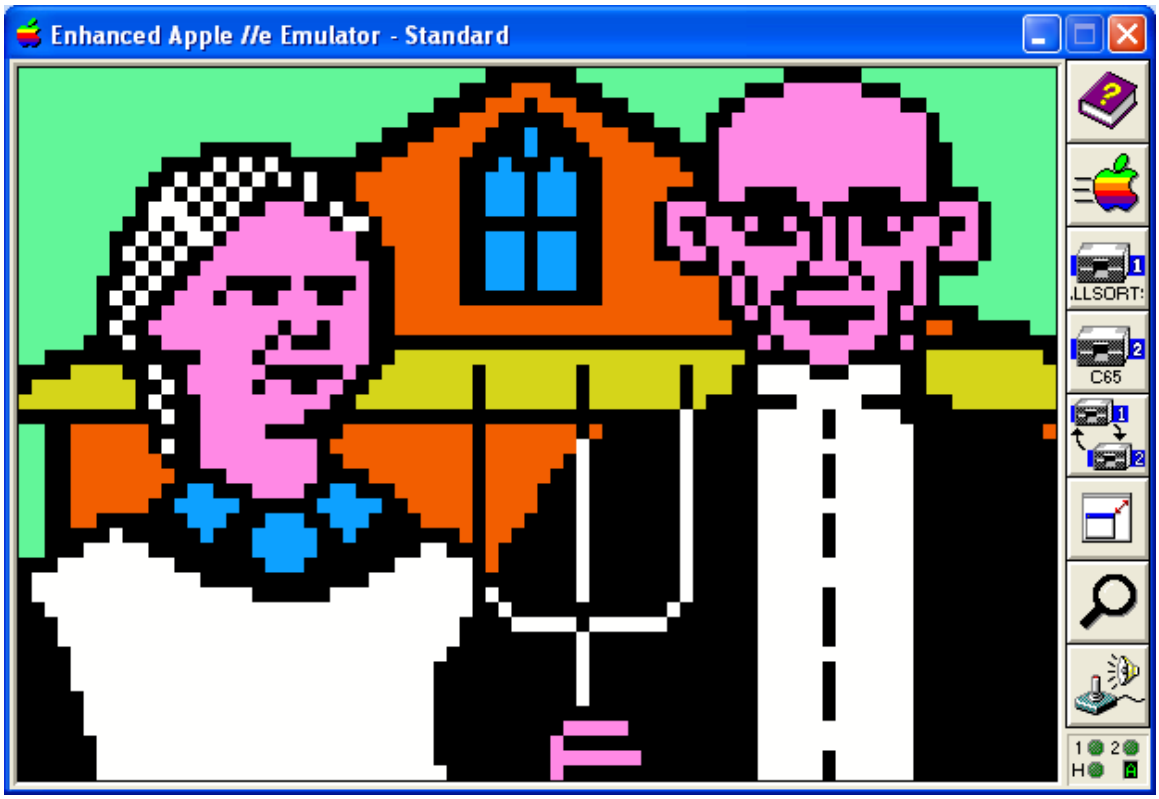
The Main Program

```
int main(void)
{
    FILE *fp;
    int status = 0;
    char picname[66];
    /* try to create a piclist if it does not exist */
    if((fp=fopen("PICLIST","r"))==NULL) {
        fp = makepiclist();
        if (fp == NULL) {
            printf("Can't open PICLIST! Press any key...");
            cgetc();
            return 0;
        }
    }
    videomode(VIDEOMODE_80COL);    /* initialize 80 column card */
    clrscr();  /* turn-on double lo-res and clear the dlgr screen */
    dloreson(); dloresclear();
/* display each dlgr pic for 5 seconds or advance on a key press. If
ESC is pressed, exit */
    for (;;) {
        while (fgets(picname,66,fp) != NULL) {
/* load picture - ignore errors */
            dlodelo(picname);
            POKE(36,0); POKE(37,0); /* HOME MESSY TEXT CURSOR */
            while (kbhit())cgetc();
/* advance on timeout or advance on a keypress */
            if (wait(5)==27) { /* if ESC is pressed, exit */
                status = -1;
                break;
            }
        }
        fclose(fp);
        if (status != 0) break;
/* no rewind in cc65 yet, so we must close and re-open the piclist when
we reach the end */
        if((fp=fopen("PICLIST","r"))==NULL) break;
    }
    dloresclear(); dloresoff(); clrscr();

    return 0;
}
```

As noted previously we use the cc65 videomode() function to assuage our pain and also to initialize the page switches that we need to flip between main and auxiliary memory for our double-res display. Using a graphics driver for a little slide-show program whether it's a BGI or a tgi driver is overkill! All we are doing is switching to 80 column mode, clearing the screen, switching to DLGR, clearing the screen, displaying slides until the ESC key is pressed, then switching to text mode, clearing the screen, and exiting. What you see above is the entire program except for the loader and a few trivial functions that are easily understood by referencing an Apple II manual. This is as dead dumb simple as an AppleSoft BASIC DLGR slide-show program☺

Not your Father's C Compiler



Graphics on any computer are not strange and frightening events happening in mysterious places but are just like pitching a little manure, and not like running a corporate farm. So even though cc65 is not your father's C compiler (Aztec C65 is), no need to fire-up the tractor and hook-up the plow, when the pitchfork will do☺

"A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. Specialization is for insects." - Robert Heinlein

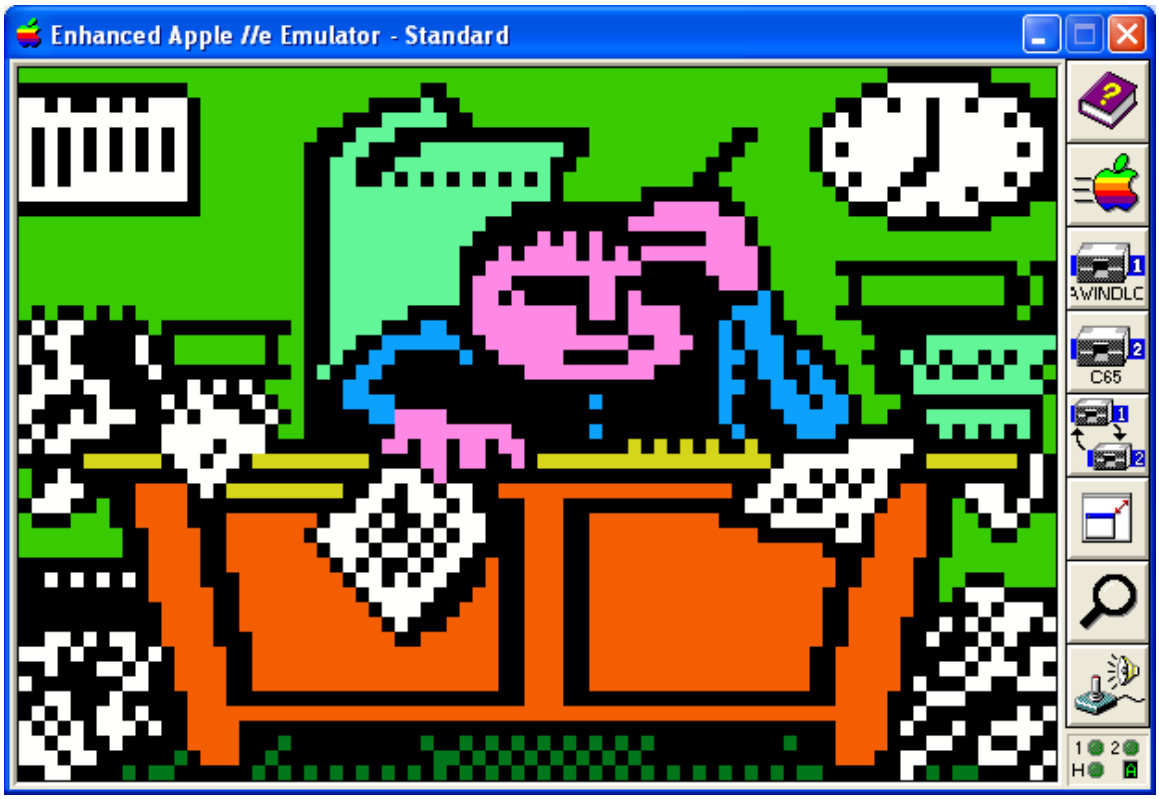
Taking the view that writing in a multi-disciplined manner is less insect-like than false inheritance and pitching manure with a plow, here's da' code:

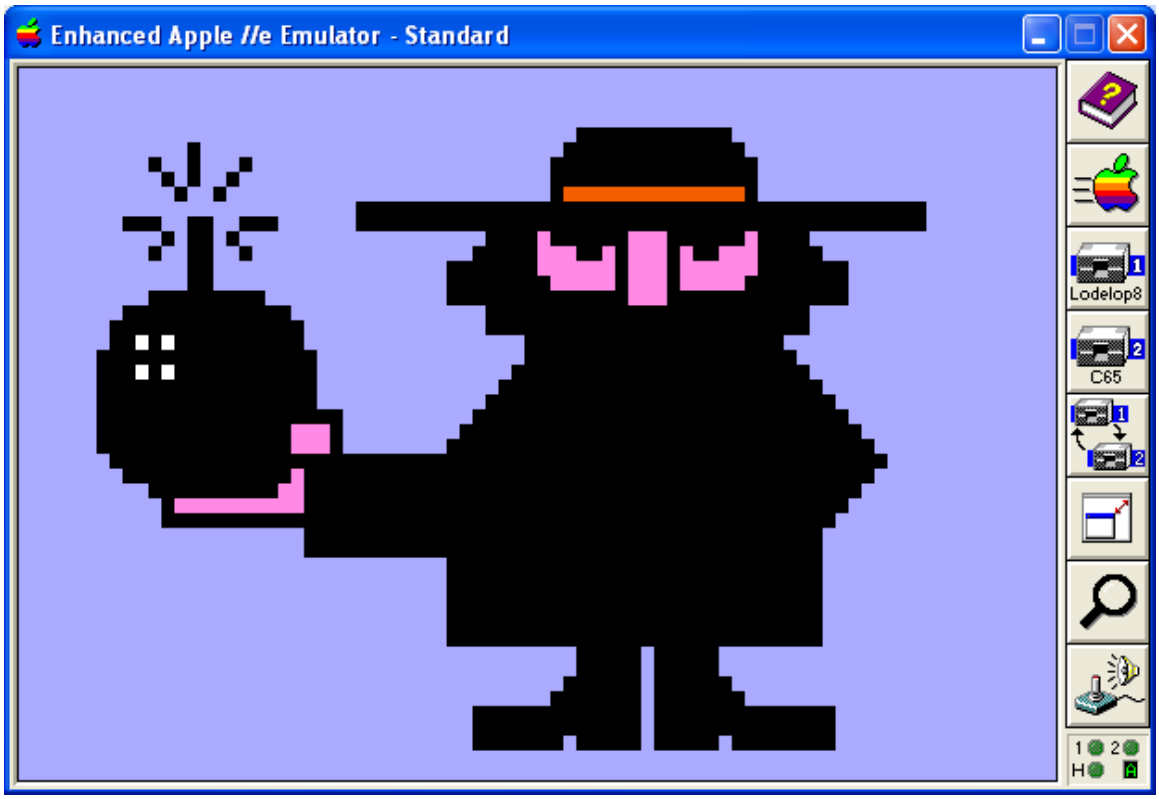
```
void dloreson(void)
{
    asm("sta  $c050"); /* GRAPHICS */
    asm("sta  $c052"); /* GRAPHICS ONLY, NOT MIXED */
    asm("sta  $c054"); /* START IN MAIN MEM */
    asm("sta  $c056"); /* LO-RES */
    asm("sta  $c05e"); /* DOUBLE */
}
```

```
void dloresclear()
{
    int idx;
    for (idx = 0; idx < 24; idx++) {
        asm("sta  $c055"); /* AUX MEM */
        memset ((char *)textbase[idx], 0, 40);
        asm("sta  $c054"); /* MAIN MEM */
        memset ((char *)textbase[idx], 0, 40);
    }
}

void dloresoff(void)
{
    asm("sta  $c051"); /* TEXT */
    asm("sta  $c05f"); /* SINGLE */
    asm("sta  $c054"); /* PAGE ONE */
}
```

That's pretty self-explanatory. You'll note a couple of things, one being the Apple II 6502 code inserted directly into the program somewhat like Aztec C65's inline assembly can be used to escape from C to do simple things (Aztec C's inline assembly can also be a replacement for full-blown assembly). The cc65 asm() statement for the built-in inline assembler is not a replacement for cc65's full-blown macro assembler which comes with the compiler. So read the cc65 documentation on the cc65 asm() statement if you want to know more (or even a little). Read the cc65 source as well.





The other thing you will notice in the above code is an array called textbase[] which simply stores the starting offsets for the DLGR screen rows:

```
/* base addresses for primary text page. also the base addresses for
the 48 scanline pairs for lores graphics mode 40 x 48 x 16 colors */
unsigned textbase[24]={
    0x0400,
    0x0480,
    0x0500,
    0x0580,
    0x0600,
    0x0680,
    0x0700,
    0x0780,
    0x0428,
    0x04A8,
    0x0528,
    0x05A8,
    0x0628,
    0x06A8,
    0x0728,
    0x07A8,
    0x0450,
    0x04D0,
    0x0550,
    0x05D0,
    0x0650,
    0x06D0,
    0x0750,
    0x07D0};
```

This array is also used when loading a DLO raster-based format. Tables aren't always good but I think this is worth the 48 bytes that it takes-up in the program to avoid calculating offsets, and the code overhead and calls etc that would be required instead. It can also be used for direct text in text or mixed mode elsewhere in a program if desired

On the Apple II some stuff sometimes gets stored in between the small gaps of the text screen. The raster format is friendly to that, but the BSAVED DL1 and DL2 files clobber that area when loaded. A programmer is always better off to use a raster approach for the reason of not clobbering memory in the text screen area even when using single lo-res. In the upper graphics area this doesn't matter as far as I know.

Thoughtful Planning

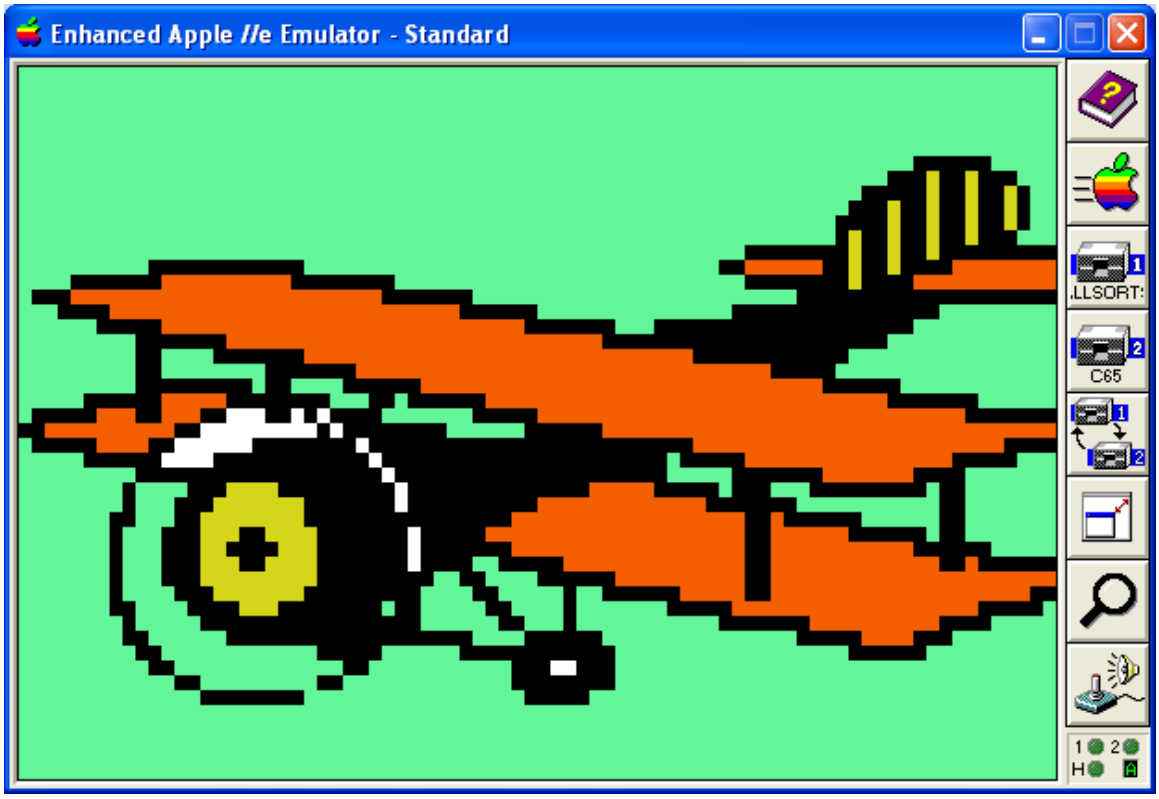
As you can see, even on a program this trivial, a considerable amount of thoughtful planning was needed to get all the pieces together. However, now that it has been done it won't take me much thought the next time, just a little cutting and pasting.

Having said that, there is no substitute for actually doing something like seeing what works and what doesn't, rather than thinking about it in terms of some library routines that someone else like Oliver or Uz have written for you. Like I said I am not a big fan of spoon-feeding of such specialized code like BGI or tgi graphics drivers. Drivers and wrappers do more to distract you than help you develop the expertise to participate in a community project like cc65. My motto is, simply put, "Work Harder"! There's no lack

of smart in those who program these old 6502 computers. But the results of my interactions with someone looking at cc65 for the thousandth time have even been hopeless, or worse, reduced to comparing performance results of printing “Hello World!” out of a C programming manual which has little to do with the Apple II or Commodore 64’s hardware. I truly cannot believe that someone like me has been able to get away with promoting an old piece of junk like Aztec C65 and amass such a volume of work for so many years while a fine modern tool like cc65 sits neglected by so many smart people. Quite frankly, I have not seen much coming out except wishes and that shouldn’t be.

So next let’s look at how easy it is to load some of these DLGR screen files. They can’t be that hard to load since even an AppleSoft BASIC program can do it☺

Loading DLGR Images – Now We’re Flying!



In the code above you will see that the asm statement is mixed with some of the other things that we have discussed like using the textbase[] array to address the DLGR screen area. You will also note that a buffer is used to read the file, rather than reading to screen memory. Reading direct to screen memory does not work in double-res mode. The buffer is required.

This isn’t something new. It happened in Aztec C65 as well. I really haven't changed this code much since writing the Aztec C65 version. All I have really done is make it fit with Oliver Schmidt's library routines as much as I felt it was necessary. I have pretty-much

written this the way a BASIC or a ML programmer who knows the Apple II might do, so folks can follow along. This is not a brilliant piece of work by any means.

```
/* The following loads two non-compressed double lo-res graphics image
formats:
   Single File Raster Format - DLO
   Bsaved Image Format file pairs - DL1 and DL2 */
char lodebuf[1920];
int dlodelo(char *name)
{
    FILE *fp;
    int y, status=-2;
    int c, fl = 1016, height, packet, jdx, idx;
    char tempchar[2], name1[20], name2[20], *ptr1, *ptr2;
    asm("sta  $c054"); /* MAIN MEM - safety play */
    jdx = 999;
    for (idx = 0; name[idx] != 0; idx++) {
        name1[idx] = name2[idx] = name[idx];
        if (name[idx] == '.') {
            name1[idx] = name2[idx] = 0;
            jdx = idx;
            break;
        }
    }
    if (jdx == 999) return status;
    /* try to open a BASIC image pair */
    strcat(name2,".DL1");
    if ((fp = fopen(name2,"rb"))== NULL) {
/* if we can't open an image pair then try for a raster file... */
        strcat(name1,".DLO");
        if ((fp = fopen(name1,"rb"))== NULL) return -1;
        fl = 1922;
        ptr1 = (char *)&lodebuf[0];
        ptr2 = (char *)&lodebuf[40];
    }
    else {
        /* if we opened a DL1 then get ready to open a DL2 */
        strcat(name1,".DL2");
    }
    switch(fl) {
        case 1922:
            /* is it a DLO ? - read the 2 byte header */
            c = fread((char *)&tempchar[0],1,2,fp);
            if (c!=2) {
                fclose(fp);
                break;
            }
            packet= (int)tempchar[0];
            height= (int)tempchar[1];
            if (height != 24 || packet != 80) {
                fclose(fp); break;
            }
            c = fread(lodebuf,1,1920,fp);
            fclose(fp);
            if (c!=1920)break;
            status = 0;
```

```
            for(y=0,idx=0;y<height;y++,idx+=80) {
                asm("sta  $c055"); /* AUX MEM */
                memcpy((char *)textbase[y],(char *)&ptr1[idx],40);
                asm("sta  $c054"); /* MAIN MEM */
                memcpy((char *)textbase[y],(char *)&ptr2[idx],40);
            }
            break;
        case 1016:
            /* is it a bsaved image ? */
            c = fread(lodebuf,1,1016,fp);
            fclose(fp);
            if (c != 1016)break;
            asm("sta  $c055"); /* AUX MEM */
            memcpy((char *)0x0400,lodebuf,1016);
            asm("sta  $c054"); /* MAIN MEM */
            if ((fp = fopen(name1,"rb"))== NULL) {
                status = -1; break;
            }
            c = fread(lodebuf,1,1016,fp);
            fclose(fp);
            if (c != 1016)break;
            memcpy((char *)0x0400,lodebuf,1016);
            status=0; break;
    }
    return status;
}
```

Additional Notes

Well, that's about it… cc65 now supports DLGR Graphics, but then it always did.

What did I really accomplish?

For one thing I produced a faster version of a program that was written in Aztec C65's old K&R dialect using modern C syntax☺

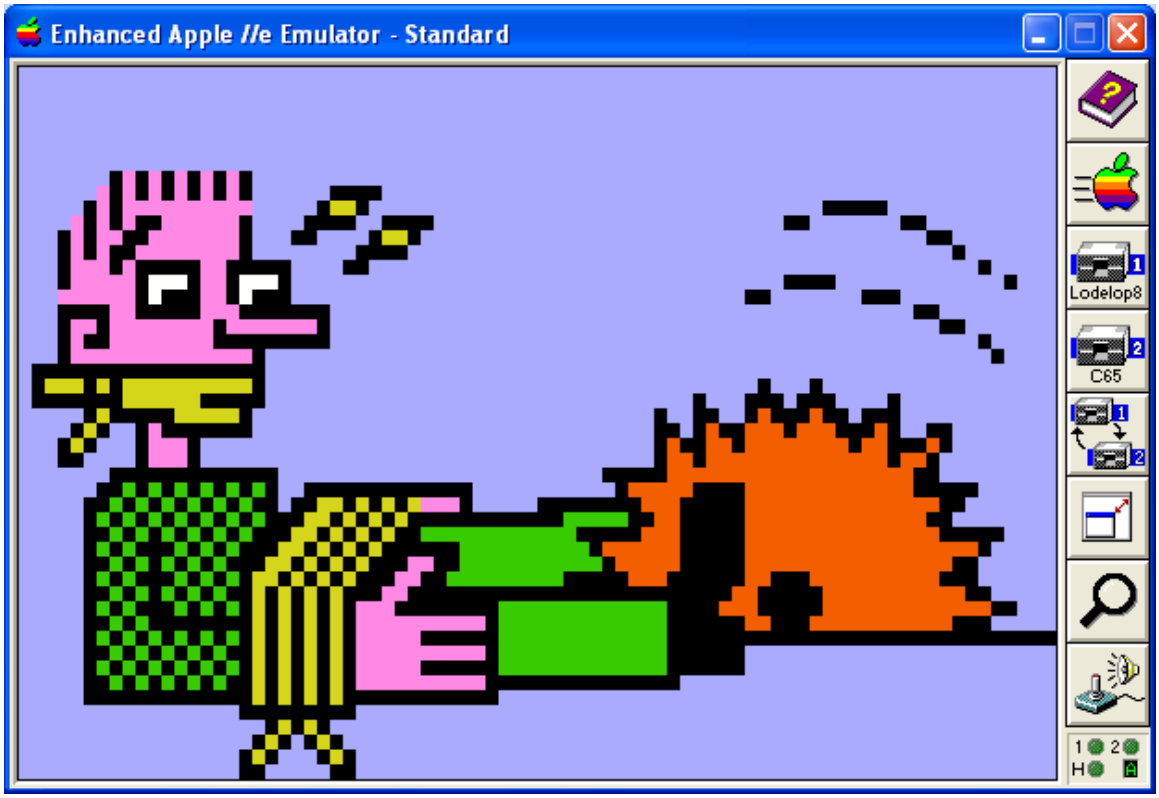
All things considered most of my time was spent poking around in Oliver and Uz's stuff to see what they were doing instead and planning a huge port, instead of paying attention to programming this demo. Actual programming and testing time was about 2 days and a 3rd for documentation, distribution, etc☺

The cc65 version itself is only 9,026 bytes and the Aztec C65 version is 25,930 bytes. The cc65 version is just 35% of the size of the Aztec C65 equivalent SYS program☺

Since all the work was pretty much done for me in this last case, this isn't really my accomplishment. I'll leave it to the reader to decide whose accomplishment it is instead of pointing more fingers from within my clown gloves. I trust all is order☺

Bill Buckels
bbuckels@mts.net

To C or not to C



The Importance of a well placed C Saw cannot be overstated.