y Jamie McCornack

Brain Drain

How nice it is to have a support group! There’s a bug in the MGW1Code package; in MGWGraphics1.c, to be specific.

Since MGW was supposed to be a training package (developed as introductory material for Tricks of the Mac Game Programming Gurus, published by Hayden Books), as a familiarizer for the basic concepts of game programming, and since this bug doesn’t manifest itself in simple programs (like the many and varied versions of HelloWorld you’ve been getting in this column), it was easy to live with, ignore, overlook, and…well, frankly, I didn’t even know it was there until readers pointed it out to me.
But since MGW is out on the streets, it is being used as the engine (to use the term very loosely) for some genuine games. This delights me, natch, except for one thing: some of the folks writing rather complex genuine games with MGW have discovered that (blush) Yours Truly wrote some significant errors into the graphics routines. Significant as in Fatal Error and Bomb Box and “the application quit unexpectedly…” alerts.

What is the bug? Let me quote some e-mail I got from Michael Andersson, who started off with some kind words like “excellent book,” before commenting re MGWGraphics1.c version 1.0.0…

“It seems there is a bug in it, though. The call to KillOffscreenPixMap() (and probably the b&w one as well) does not release the memory properly. Creating and removing a lot of offscreens will fill the available memory pretty quickly.”
Well, that’s pretty straightforward. It’s sure a lot easier to understand than a comment written with THE CAPS LOCK KEY HELD DOWN AND LOTS OF EXCLAMATION POINTS!!!!!!!, not to mention distracting references re my mom (wow, I never knew she did that, but nevertheless, let’s get back to the subject of game programming), but hey, it doesn’t sound too bad. “Available memory” is a relative term, and some quite decent games have modest memory requirements. Dark Castle (unsolicited plug: now available in color from Delta Tao) runs great on a 1-meg Mac Plus, and though my own wetware memory is losing reliability as the decades flow, I seem to recall the original Lode Runner running on a 32K Apple II. So what’s the big deal about not releasing memory properly?

Let’s look at what KillOffscreenPixMap() is supposed to do.

CreateOffscreenPixMap() allocates memory as needed for the size and depth of the desired offscreen pixel map. If you want to make a pixel map that is 640 by 480, you’d pass CreateOffscreenPixMap() a rect parameter of that size, which is 307,200 individual pixels (640 times 480), and if you’re running 8-bit color (eight bits is one byte) your pixmap requires about 30K bytes of RAM. No big deal, right?
When your program is done with that particular pixmap, calling KillOffscreenPixMap() frees that memory for further use. Hah! It’s supposed to, but it doesn’t!

You could go years without this being a problem, if all your backgrounds for all your levels were the same size, presuming you used one offscreen workMap and one offscreen backgroundMap, and instead of killing the old maps and allocating new maps with each level change, you kept the maps and used CopyBits() to replace the previous level’s data with the new level’s data. Or you could use a single background and change levels by placing new platforms, images, etceteras, against a common background, a la john calhoun’s Glypha III. Those of you who have Glypha III source code (a fully commented version comes on the Tricks of the Mac Game Programming Gurus CD, and there’s a standard version floating around on various on-line services) will note that while it includes a CreateOffscreenPixMap() routine (which I pretty much used word-for-word in MGWGraphics1.c, which is one reason john’s name is in the copyright notice), it doesn’t bother with KillOffscreenPixMap() at all. All Glypha III’s pixmaps are used throughout the game, and they get “killed” when you exit the program.

You, the programmer, don’t have to worry much about where the memory goes after the user hits Quit. The MacOS will take care of freeing up memory after your program is gone. But how memory is used while your program is running, well, that may require your attention.

It’s time to quote another reader, Kevin Teich this time. I’m going to quote Kevin at length because his analysis and solution are pretty right on—better than mine, for sure—so you can think of him as a guest author for this month’s column.

“…messing around with the MGWGraphics1.c file in chapter 0, attempting to adapt it to one of my own programs...[which] uses 16 source pictures to compile a composite picture to draw onto a window. Before, I had it set up as 16 GWorlds and a buffer GWorld for the composite picture...anyway though, I was attempting to change it to 16 pixmaps instead…in using CreateOffscreenPixMap and KillOffscreenPixMap to make and dispose of the 16 pixmaps, I noticed a serious memory leak. In loading a new map, the 16 pixmaps are disposed of and reallocated with new picture data, [and the routine] that Killed them wasn't disposing of them correctly.”

Okay, if you’re using 16 offscreen pixmaps as source pictures, and replacing all 16 with new pixmaps at each level change, you’re probably going to notice something is wrong. Kevin didn’t mention how much time he spent screaming and tearing handfuls of hair out of his head, but I’d suspect some of that went on before he concluded that KillOffscreenPixMap() wasn’t performing as advertised.

However, it could have been much worse. What if he’d only been using eight offscreen pixmaps, and two or three levels? He might have had a sneaky memory leak problem instead of an obvious one; a memory leak that almost overflowed the stack, but not quite, so that something else might have seemed to be the culprit.

For example, it might not have left enough memory for sound resources. This is hypothetical, but imagine a set of ‘snd ’ resources that include a 200K music loop, a bunch of 10K to 20K clanks and yelps, and a 40K death moan. The game starts, and up comes the splash screen with the music loop playing in the background, and every thing seems groovy. When the game itself begins, the pixmap for the splash screen is disposed of (hah, but it’s not, because of the bug, so some memory leaks away), and the game begins, with the player’s character clanking and yelping and even death moaning when appropriate (you get three lives in this game), working up to Level 5, by which time the bug in KillOffscreenPixMap() has leaked off all but 30K of the available RAM.

Now there are little random pauses during the clanking and yelping parts of the game, because sometimes, when a clank is called for, the memory manager has to purge a yelp, and then access the hard drive to load the clank snd resource into the RAM recently vacated by said yelp. Kind of mysterious, eh? Must be a sound problem.
And when the player’s character is killed, as soon as the death blow is dealt, the game crashes. Another mystery. This didn’t happen on Level 4, I wonder what we did wrong on Level 5?

End of hypothetical sidebar. Back to Kevin, who had the good fortune of a real straightforward memory leak problem, which he traced back to MGWGraphics1.c.
“Using zone ranger, I figured that each Create allocated 78304 bytes. Kill only freed 1072. So I figured out that CloseCPort doesn't dispose of the HandToHanded CTable, which is what is done in Create. Adding in a DisposeCTable call to Kill freed another 2144, but that still left 75000 unaccounted for. my pixmap is 300 pixels by 250 pixels - doing your rowbytes calculation, I figured out this was the amount of memory allocated to the pixmap in Create. So, my guess is that Kill isn't disposing the pixmap properly.”

Brilliant, Holmes. So I started up Zone Ranger from the Other Metrowerks Tools folder in my CodeWarrior folder, and by golly, Kevin was right!

So we give Inside Mac: Imaging a look, and find that CloseCPort, “...also disposes of the graphic port's pixel map, but it doesn't dispose of the pixel map's color table…” which explains why the color table is still in memory, but why isn’t the pixel map gone? Beats me. The data may be gone, but the memory lingers on. CloseCPort releases memory used in the visRgn, clipRgn, bkPixPat, pnPixPat, fillPixPat, and grafVars but doesn't do diddly to the portPixMap, not as far as I can see. We’ll call it One of the Mac’s Many Mysteries for the moment.

So here’s the fix, or at least <a> fix. In version 1.0.0, that routine used to read like this:

```
void KillOffscreenPixMap (CGrafPtr *wasPort)


 if (wasPort != nil)

     CloseCPort(*wasPort);
     *wasPort = nil;
```

The New and Improved version is as follows:

```
void KillOffscreenPixMap (CGrafPtr *wasPort)



 if (*wasPort != nil)

      // Release the data in wasPort.portPixMap
      // since CloseCPort won't do it for us.
     DisposePtr(((**(*wasPort)->portPixMap)).baseAddr);
      // Get rid of its ColorTable since we gave
      // it its own separate copy.
     DisposeCTable((*(*wasPort)->portPixMap)->pmTable);
      // And then, close wasPort.
     CloseCPort(*wasPort);
     *wasPort = nil;
```

The black&white single-bit routine got a similar treatment (and remember, you have to use a bitmap if you want to use CopyMask()), like so:

```
void KillOffscreenBitMap (GrafPtr *wasPort)



 if (wasPort != nil)

      // Release the data in wasPort.portPixMap since
      // ClosePort won't do it for us.
     DisposePtr(((*wasPort)->portBits).baseAddr);
      // And then, close wasPort.
     ClosePort(*wasPort);
     *wasPort = nil;
```

There, that wasn’t so bad. Memory leaks plugged, you can Create and Kill offscreen pixmaps and bitmaps ‘till the cows come home, and we’re all happy campers.
You’ll find a new MGW1Code folder, containing the improved graphics file, in the folder named MGD4/96. The file is still named MGWGraphics1.c, but you’ll note in its heading that it’s now version 1.0.1, and I’ve added Kevin Teich’s name to the usual suspects list.

So tell me, what would you like to see in MGW2Code? Yes, a PPC Native version would be nice, I think all it would need there would be to change the callback routine in MGWSound1.c. Anything else? If you have a wish list, drop me a line at MacGameDev on AOL (that’s <macgamedev@aol.com> for   other service users) and we’ll talk it over.

I’ve also included a Universal Clam Resource (UCR) which you can use with any of the color HelloWorld projects (including GoodbyeWorld from the February ‘96 IMG, though you’ll have to change the GoodbyeWorldSnd resource ID# to 3010 instead of 3000), or mix and match to create your own. HelloWorld6.π is included for those of you with CodeWarrior 6 or later, including the CodeWarrior Lite versions, but who tuned in late to this column. See you next month! I mean, SEE YOU NEXT MONTH!!!!