and then displayed, alternately between each cycle. The eye is fooled into seeing a stable target all the time, with the phaser fire emanating from it. It also gives the phaser its quick, pulsating effect. The target is erased first so the routine returns with the target on, though other variations are possible.

Looking closely at the routine, you'll see the redundancy of the display and XY coordinate adjustments. I wrote four versions of this routine, searching for a more efficient loop, but was only able to cut two lines (including the bytes for the patterns) by converting the redundancies to a subroutine. Since the three subroutine calls (and three returns) added more time to the loop than I saved by cutting two lines, the redundant code stayed. Losing the fast pulse of the target's phaser fire would detract from the game, I felt. I could have displayed and erased the ship more often for less of the "blinking" effect, but again, the speed would suffer. In this version, the target stays on quite long enough to fool the eye, but for more complex transparent graphics, you'll have to have many more DXYN instructions to get the desired effect.

Next, and in the same routine, the program uses the same random decision technique to decide if your starship has been hit. Though generating a random number from 00-07 will produce zero 12.5% of the time, the routine itself is called randomly. Therefore, your starship is hit only a percentage (12.5%) of the time that the program decides to allow the target to fire (3.12% of the time). So your chances of being hit are really only 0.25% (3.125 ÷ 12.5) of the times the program passes through the main loop. This is a delicate balance which seems to produce very competitive results. (That means the target hits you about the same number of times you're able to hit it.) By all means, play around with some