trail bip to allow for erasing its "behind" and for resetting if (when!) you run into the wall.

Continuing at 0258, the next trail bip is displayed, a tone is sounded, and a test is made to determine if the trail hit anything (if VF=1). If so, control passes to the scoring routines at 02CA, which determine if a target was hit, check for a win, display the score, etc....More on this routine a little later (when we get to it in the natural sequence of things).

If the trail didn't hit anything, the next section determines the speed of the trail. (Line 0264 is a "no operation" instruction which replaces a "random speed increase" that didn't work out very well.) Every time the trail cycles through one 80-bip distance (indicated by V7=9E), the speed is increased by subtracting 2 from V5. The maximum speed that the trail can attain is set in line 026A, which may be changed to 35KK, in which KK is any even number in the range 00-0C. Try 3504 or 3506 if you think the trail moves too fast; if you try 3500, the trail will eventually shift into full warp drive.

Lines 026E and 0270 demonstrate the use of a helpful timing techniques I use in all my game programming. Turn to line 038C. You'll see a simple timing subroutine that sets the CHIP-8 timer equal to the value of VE, tests to see if the timer equals zero yet, and only returns via an 00EE instruction afterwards. (The timer, remember, automatically begins to decrement the instant it is set, so the same variable (VE in this case) can be used to see when it goes to zero, so long as the value doesn't have to be preserved.) I use this subroutine in many ways - which is a good reason to have it in subroutine form - and only have to set a value into VE, call the routine with a 2MMM instruction, and the program will halt where it is for a value relative to VE!

Getting back to 026E: You'll see that VE is set equal to V5 (the trail-speed indicator) and thetiming subroutine is called.