We now have a reasonably well-developed point-placing subroutine:

```
330 REM PUT THE CHARACTER CS AT X(OVER), Y(DOWN)
340 IF X > 80 OR X < 0 OR Y < 1 OR Y > 24 THEN RETURN
350 IF X = 80 AND Y = 24 THEN X = 79
360 LOCATE Y,X : PRINT CS
370 RETURN
```

Of course, the tests in lines 340-350 can be eliminated if the program tests those conditions before calling the subroutine.

On my way up from the bottom, I asked: can I draw a line? Recalling my analytic geometry, here is a first draft of a way to draw a line from X1,Y1 to X2,Y2, marking each point with character CS:

```
390 REM DRAW A LINE FROM X1,Y1 TO X2,Y2: ASSUME
400 FOR Y = Y1 TO Y2
410 X = (2X1-X2)/(Y2-Y1) * (Y-Y1) + X1
420 GOSUB 330
430 NEXT Y
440 RETURN
```

A quick experiment reveals that that does not work satisfactorily; X is rarely an integer. Adding \( 415 X = INT(X + .5) \) helps, but the loop is still much slower than it should be: we are doing unnecessary repeated calculations during the loop. Here is a better version:

```
390 REM DRAW A LINE FROM X1,Y1 TO X2,Y2 WITH
400 FOR Y = Y1 TO Y2
410 S0 = (X2-X1)/(Y2-Y1): S =X1-S0
420 FOR Y = Y1 TO Y2
430 S = S + S0 : X = INT(S + .5) : GOSUB 330
440 NEXT Y
450 RETURN
```

This version makes fewer calculations inside the loop.

We could have another version of this program in case \( Y = Y2 \), interchanging \( X \) and \( Y \) throughout, and maybe even a test for whether \( ABS(X1-X2) > ABS(Y1-Y2) \): do we want a line moving nearly horizontally to show as \( XXXXXX \) or as \( X \quad X \quad X \quad ? \)

Since the latter will give faster motion, I settled for it. On that basis, the routine just given will plot any descending line.
These commands and details particular to the IBM Personal Computer are provided so that users of other microcomputers can substitute as may be required by their systems:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLS</td>
<td>Clear Screen. This applies only to lines 1 - 24 unless KEY OFF is in effect.</td>
</tr>
<tr>
<td>KEY OFF</td>
<td>Turns off and on the line 25 display of meanings of the 10 programmable keys.</td>
</tr>
<tr>
<td>KEY ON</td>
<td>Moves the cursor to line 1 range 1-25), position B (range 1-80).</td>
</tr>
<tr>
<td>LOCATE A,B</td>
<td>Returns an integer: the number of the character presently appearing at line A, position B, on the screen.</td>
</tr>
<tr>
<td>SCREEN A,B</td>
<td>Returns a string of N characters long, each character is character number K.</td>
</tr>
<tr>
<td>STRING N,K</td>
<td>Sounds the speaker, frequency F, for duration B units.</td>
</tr>
<tr>
<td>KEY IN</td>
<td>The one or two character long character string denoting the most recently pressed key, or the empty string if no key has been pressed since it was last referenced.</td>
</tr>
<tr>
<td>RANDOMIZE N</td>
<td>Restart random number sequence, based on the seed N. An unpredictable N may be obtained by extracting substrings from TIMES.</td>
</tr>
<tr>
<td>TIMES</td>
<td>The clock time since the system was booted up, as a character string, of the form 02:25:14 (for 2 hours, 25 minutes, 14 seconds. May be reset to actual time, if desired.</td>
</tr>
</tbody>
</table>

Screen Characters used:
- 2 Bright Face (1 is Dark Face)
- 25 Down arrow,
- 219 Full box (all white if writing white on black)
- 178 Shaded box (grey, if writing white on black)

Keyboard Characters used:
- 13 Enter, often called carriage return.
- 32 Space or blank.
- 0-77 Character right returns a two-character string,CHR$(0)+CHR$(77)
- 0-75 Cursor left
- 0-72 Cursor up
- 0-80 Cursor down
- 0-83 DEL (Delete)
- 0-82 INS (Insert)

A Top-Down Approach
With those subroutines in hand, I soon had a slow shower of characters falling down my CRT screen. A game began to take shape in my mind.

A Note On RANDOMIZE
IBM Personal Computer Basic will produce the same sequence of random numbers each time unless you use the command RANDOMIZE. You must provide a “seed,” or starting value, in the range -32767 to 32767. Of course, you would like that seed to be unpredictable, and if possible different almost every time you start. Here are two methods:

- In this game there is a keyboard input called for very early: DO YOU WANT DIRECTIONS? In a loop at lines 160-170, we repeatedly tested (using INKEY$) to see if the Y, N, or Exit had been struck. The variable R is incremented each time we go around that loop, and kept in the range 0 to 32003 by the MOD (modular arithmetic) addition. Thus the seed in RANDOMIZE R depends on how quickly the user pressed the key.
- A somewhat easier method is available in Disk Basic or Advanced Basic, since these have the keyword TIMES. One and a half minutes after the computer has been turned on, if the user has not reset this variable, it has the value “00-01:30” (No hours, one minute, thirty seconds). We can convert it to a number and use it as a seed by a command such as RANDOMIZE VAL(MID$(TIMES,7,2)+MIDS (TIMES,4,2)). This will produce RANDOMIZE 301 at the just-mentioned time after startup. While this method is certainly easier than the first method, I have included the first method in the listing since the second method is unavailable in cassette Basic.

Trial Program Outline (Version 1)
2. Create a marker showing where the “player” is.
3. Periodically have objects falling from the top of the screen.
4. Give the player a way to move his marker, to dodge the falling objects.
5. Turn ends when the player’s marker is hit by a falling object.
   This was not entirely satisfactory. It would be far more satisfactory if there were something the player could do (other than simply survive) to score points.
   Basic character graphics run a bit slowly to allow shooting sorts of games, which I don’t really like very much anyway. I decided instead to try letting the player marker move through a path or obstacle course, or gather points when it got to certain targets.

Program Outline (Version 2)
2. Create targets on screen.
3. Choose a path for a falling “meteor.”
   Each time the meteor falls one position:
   - If it hits the player marker, go to step 13 (end of turn).
   - Erase old meteor position, mark new one.
   - Does player want to move his marker? (Read keyboard)
     If he does:
     - If new position is occupied, perform step 12 (score).
     - Erase old position, mark new one.
   - If meteor has further to fall, go to step 4.
   - Return to step 3.
   - (Score) Depending on the target hit, increment score, display score on the screen, make a noise: return to main program.
4. (Player is hit). Sound a noise. Draw explosion. Await input to decide whether to play again (go to step 1) or exit program.

August 1982. "Creative Computing"
Here are some pictures showing the game Meteor being played. The pictures were made using an Epson MX-80 printer, which requires a somewhat different character set than is used on the CRT screen. The meteor trails are shown here as short vertical marks (|) and the player's position is marked with a ®.

**Meteor**

A simple arcade game using character graphics.

The cursor control keys start the ¢ function, moving the space bar stops all action temporarily, and allows restoring targets.

**Early in the game.**

See if you can erase the solid blocks before a falling meteor hits you. Each time you erase scores 10 points. Each 2 points, you lose 1 point for each 3 a meteor hits.

To hit you a meteor needs to get within the shaded area.

Some extra instructions will be on the bottom line.

How many (1-9)?

The directions as shown on the screen (symbols altered for printer).

**Converting Graphics for Printer Output**

I have an unmodified Epson MX-80 printer. Its graphics characters are in many ways better suited to the TRS-80 than to the IBM Personal Computer. Still, I wanted to print enough to show what the screen looked like while playing the game. Accordingly, to make the screen printouts provided with this article, I did the following:

For the characters used in the program, I substituted characters that would have a similar general visual effect on the Epson printer. This involved changing lines 120, 140, 370, 700, and 710. The substitutions I made were as follows:

For CHR$(219)" solid square" I used CHR$(223), the printer solid block.

For CHR$(25)" down arrow" I used CHR$(124), the vertical line 1.

For CHR$(178)" shaded square" I used CHR$(61), the equal sign =.

Then I set the printer in compressed character (132 characters per line) mode to make the printed shape close to the original screen shape. I ran the game, periodically using the pause-on-space-bar feature of the game to halt the action, and print out the screen using the built-in Print Screen key.
Two Minor Nuisances

I encountered two minor nuisances that I would regard as slight criticisms of the hardware and software design for the IBM Personal Computer.

• The cursor control keys double as the numeric keypad; the Numeric Lock switch alters their function. If you accidentally strike this key during repeated use of this keypad, strange things happen. In the case of this game, when you are striking cursor control keys repeatedly, hitting Numeric Lock freezes the player marker on the board and inhibits response to the cursor control keys. (This could be overcome in the game design.)

• When debugging, I frequently LIST a program and stop the scrolling to read a portion of the listing; scrolling is stopped on the IBM Personal Computer by Control-Numeric Lock. Unfortunately, this can freeze the screen in the middle of a single scroll, causing one line of the screen (the one being rewritten) to contain false information about the program contents. (The screen will regenerate, correcting the error, if you try to exit the listing to fix the error.)

390-430 could change the character of motion of the attackers or meteors. It would be quite easy to install obstacles, that is, points through which the player marker, meteors, or both could not pass; testing that the player marker is not hitting such an obstacle would go in at around lines 700-720.

A description of the code, with references to the Program Outline (Version 2) above, follows:

Line 120 clears the screen and initializes.
Line 150 offers directions. If wanted, GOSUB 930.
Line 180 randomizes. IBM Basic requires a seed; see box for discussion.
Line 190 asks how hard it should be (move targets up or down screen). Note: Hitting Enter alone causes defaults: no directions. difficulty = 5.
Line 230 chooses starting location HX, HY for the player markers.
Line 240 clears the screen (step 1 of outline).
Line 250 places instructions in the bottom line.
Line 260 uses GOSUB 840 to place targets on the screen (step 2).
Lines 270-320 choose a path for the falling meteor (step 3, 11); GOSUB 390 to actually plot the line.
Line 330 (subroutine) plots a point for the meteor; checks for player marker hit and for keyboard input. If HS is not null, but KS is, continue player marker motion as before (step 5-6-7).
Line 390 (subroutine) draws a line for the falling meteor. If the meteor is below the player marker and cannot hit it, terminate line (step 4).
In line 450, the player is hit by a meteor. The program waits for the player to hit the INS(err) key to restart or, the DEL(elete) key to exit (step 13).
Lines 570-730 process requests for player marker movement (step 8-9).
Line 740 increments score (step 12).
Line 760 processes pauses resulting from depression of space bar. It offers choices of continuing, exiting program, restoring targets.
Line 840 prints targets and places player marker on screen.
Line 930 gives directions.

"Might as well cancel your ad; today, all the printing is done electronically."

Meteor, continued...

Built-In Machine Functions

It is now clear that we need to know at least two more functions: how to find out what is displayed at a point on the screen, and how to detect keyboard input without waiting for an INPUT statement.

The first function is easy: SCREEN(Y, X) is a built-in numeric function which returns the number of the character present on the screen in position X of line Y. (In some versions of Basic, on other computers, this may take a PEEK or other technique. In extreme cases—a computer talking to a very dumb terminal—it may require keeping a copy of the screen in an array in memory.)

The second is also easy: INKEYS is a reserved word whose value is the key recently pressed. There is a slight complication, however: on the IBM Personal Computer, some keys produce a two-character value for INKEYS. For example, the Home button produces CHR$(0)+CHR$(71) for INKEYS.

From here on, it is largely a matter of picking specific characters for attractive graphics, fine tuning, and "dressing up" the game. The IBM Personal Computer provides a nice range of graphics symbols, including two small faces, one of which I selected for the player marker, and a downward arrow which was ideal for a falling meteor.

To create more scoring activity, I deducted a point when a meteor hit a target, and left meteor trails on the screen, awarding points when the player erased them. I put in a pause feature controlled by the space bar and a provision for restoring the targets when they were all erased.

Survey of the Code

A listing of the program is provided in Listing 1. Since the code, including directions and remarks, is under 100 lines, it should be easy to alter. The relatively modular design makes it easy to change into games having little superficial resemblance to the one shown here. For example, changing lines 840-900 would rearrange the targets; an alteration in lines 280-310 or
Learn to type with HiRes Typing Tutor

Seventeen exciting keyboard lessons in a colorful, fast-moving game format. For Apple II* or Atari 800* with 48K and disk. Just $39.95 from your favorite dealer, or write:

Lightning Software
P.O. Box 11725, Palo Alto, CA 94306
415/327-3280.

August 1982 * Creative Computing