ODD MAN AT PLAY

Transforming a concept into a game

by JOHN and MARY HARRISON

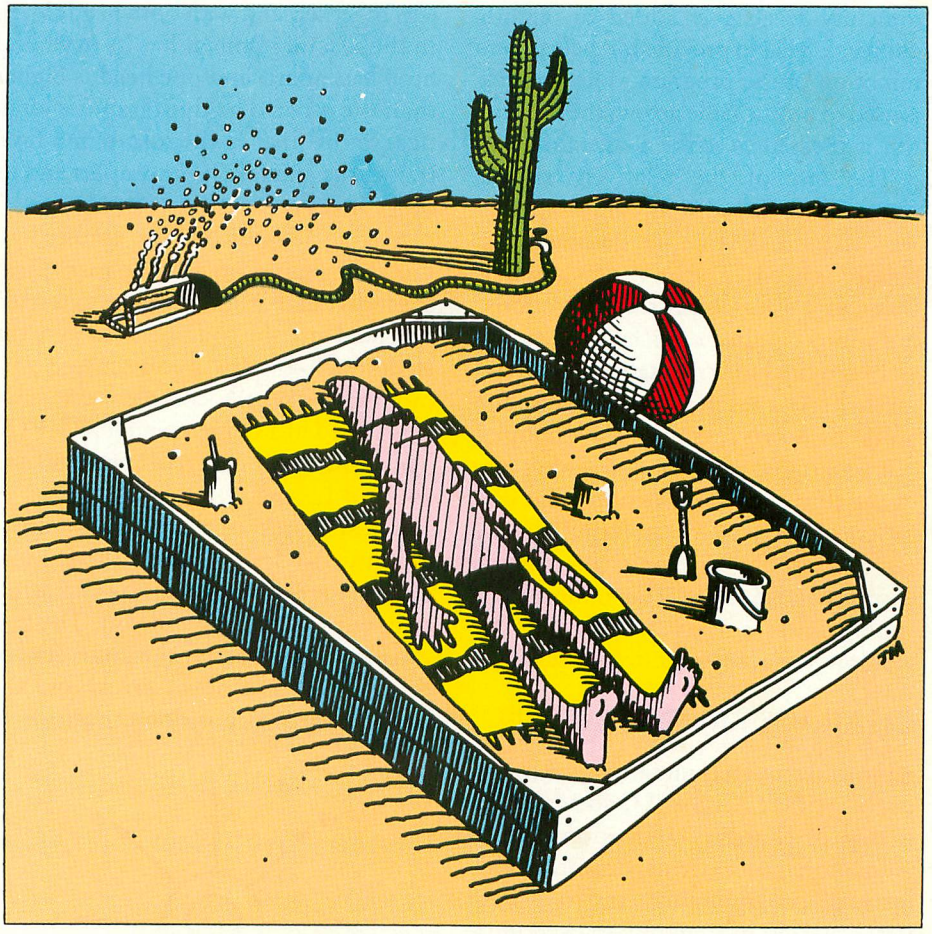
Last month we developed the custom character set that is the backbone of **Odd Man Out**, an educational program we designed to develop visual discrimination skills in preschoolers. This month, we will examine the subject of *play action*, the interaction between computer and user that transforms a concept into a game, as part of our study of how to write educational programs for preschool children.

THE BASICS OF PLAY ACTION

No matter how well conceived an idea seems to be, it is not effective unless it is used. Because of this, one of the most important facets to consider in designing educational software is play action. In developing Odd Man Out, we considered the following elements:

- varying background colors with difficulty level
- choice of graphics mode
- making the joystick responsive for young children
- adding an animated character to use in selecting proper response
- random selection of displayed objects
- immediate reward for correct response
- immediate indication of incorrect responses (without being critical)
- graphic reward based on performance

Some of these items are quite simple and require little or no explanation, but it is well worth the time it takes to examine the implementation of the more complex elements.



IT'S THE SIMPLE THINGS THAT COUNT

The pre-school audience lacks the sophistication of older and more computer-aware age groups. The three or four year old doesn't care if the animation is jerky or the graphics are simple — if he can gain mastery over the computer and make it perform for him. While this should not be used as an excuse for shoddy work, it's true that elaborate or highly-detailed graphics displays are rarely appreciated by the preschooler. Often it is the simple, subtle touches that make a program for youngsters easy to use and interesting to play.

By varying the background color with each difficulty level, we can immediately cue both parent and child to the level of

difficulty being attempted. This is simply accomplished. In Odd Man Out, the variable CHOICE designates which of the seven levels has been chosen. Since the Atari is capable of displaying sixteen basic colors, we multiplied CHOICE by two and used the SETCOLOR command to change the background color. Color selection is arbitrary, but we preferred to use the even-numbered color choices.

To make the menu appealing, we used Graphics Mode 1 which contains large characters and thus makes the menu easy to read. An arrow points to the current selection. In addition to the seven difficulty levels, we added an eighth choice, EXIT, to provide for a clean termination of the program. The joystick is used to position the arrow; the fire button is pressed to make a selection.

The last of the relatively simple elements we included in Odd Man Out was a joystick routine designed for children's needs. The joystick is a switch that can sense eight distinct directions, but young children often lack the motor skills required to accurately position the joystick. As a result, when it checks for an up or down response, for example, our program checks for and allows any of the three joystick positions that can be considered to be pointing down, rather than allowing only one correct choice (see Figure 1). This minimizes a young child's anxiety by widening the window for acceptable responses.

CHARACTER ANIMATION

A more obvious technique we used to improve the Play Action of Odd Man Out is that of character animation. While animation is an extremely complex element of programming, and one that usually results in crude and jerky motion in BASIC, our implementation of animation in Odd Man Out is fairly simple and yields smooth motion.

Because this technique only works in Graphics Mode 0, we used the text window at the bottom of the Graphics Mode 2 display as the background for our animation. The Graphics-0 characters are small, so we designed an object three-characters-wide and two-characters-high. We wanted to be able to move the object both to the left and the right, and to keep its implementation simple. As a result, a two-headed animal seemed appropriate (our apologies to Dr. Doolittle). A child can position our "pushme-pullyou" beneath the object he or she wishes to choose without facing the distraction of seeing an animal that appears to walk backwards.

We started this process by attempting to visualize how such an animal would move. We decided that four "frames" would be required to describe the motion of the pushme-pullyou. In the first frame, the animal stands with its front and hind legs together, and with both heads level. In the second frame, it has its front and hind legs apart, and one head is higher than the other. The third frame is identical to the first. In the fourth and final frame, the animal's legs are apart and its other head is raised.

Having identified these frames, we drew the figures on graph paper. Then the individual characters were extracted and read into the computer (as were the custom characters we discussed last month). Next, the complete sequence was stored in a string variable, PMPY$. This sequence is shown in lines 1380 and 1400 of Listing 1.

These lines appear to be quite obscure, so let's look at them more closely. Remember it takes six characters to define the pushme-pullyou. The first three characters define the upper half of the animal. The down arrow and the three left arrows position the cursor so that the bottom half of the pushme-pullyou is drawn beneath the upper half. This completes the first frame of the sequence. The rest of PMPY$ contains the remaining three frames of the animated sequence.

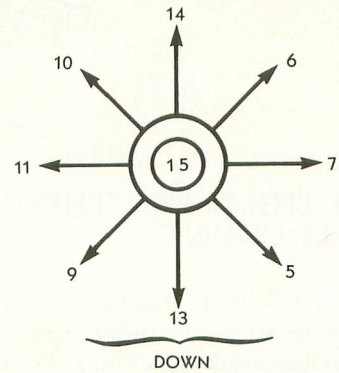


Figure 1

Odd Man Out allows for more than one correct choice of direction on the joystick.

To prevent the danger of having bits and pieces of the pushme-pullyou strewn randomly about the screen, a mechanism is needed that erases those portions of the screen that are no longer occupied by the animal. This is accomplished by means of the variable ERASE$. Thus, we print ERASE$, the frame, and then ERASE$ again. This allows the animal to move to either the left or the right without leaving a trail of abandoned parts behind.

RANDOM DISPLAYS

Nothing makes a game more boring than being able to foresee what sequence of problems or obstacles will be displayed. We have attempted to avoid this pitfall in Odd Man Out by using the random-number feature of Atari BASIC. We use this random feature in three ways. First, we randomly select the object that will be displayed three times in a particular problem. Second, we randomly

continued on next page

John and Mary Harrison are parents, teachers and Atari hobbyists. Mary teaches math and computer science at the high school level. John holds an M.S. in computer science and develops educational software. They coordinate the Education Department for ANTIC.

"Odd Man At Play" is the third installment of a series designed to teach the basics of writing educational software. The series is based on a program called Odd Man Out. Part of the program listing appeared in the December issue; the listing and its explanation will be completed in February. As a result, those portions of the program that appeared in the December issue and that appear in this issue do not constitute a complete program. The system requirements of the completed program will be either 24K RAM (cassette) or 32K RAM (disk). —ANTIC ED

choose the object that will be displayed only once (the odd man out). Third, we randomly select the position that will contain the odd object. These steps ensure that there is no bias in our program regarding either the objects to be displayed or the location of the odd object in each set of objects.

EXPLANATIONS OF THIS MONTH'S PROGRAM SEGMENT

This month's program segment consists of the following major sections:

(1) Lines 140–560: declare variables and read in characters for the pushme-pullyou
(2) Lines 880–1300: select difficulty level
(3) Lines 1320–1420: define frame sequence for character animation
(4) Lines 1500–1860: create level 1, 2, and 3 displays
(5) Lines 1880–2200: create level 4 and 5 displays
(6) Lines 2220–2480: create level 6 and 7 displays
(7) Lines 2500–2860: joystick input and pushme-pullyou move routines
(8) Lines 4180–4420: data that defines the pushme-pullyou

The first section requires little explanation. The variable FRAMES is the number of animation frames used (in this case four). FRAMSZE holds the length of a particular frame. In our case, ten characters are required per frame (six for the pushme-pullyou, along with four cursor-positioning characters used to properly align the object).

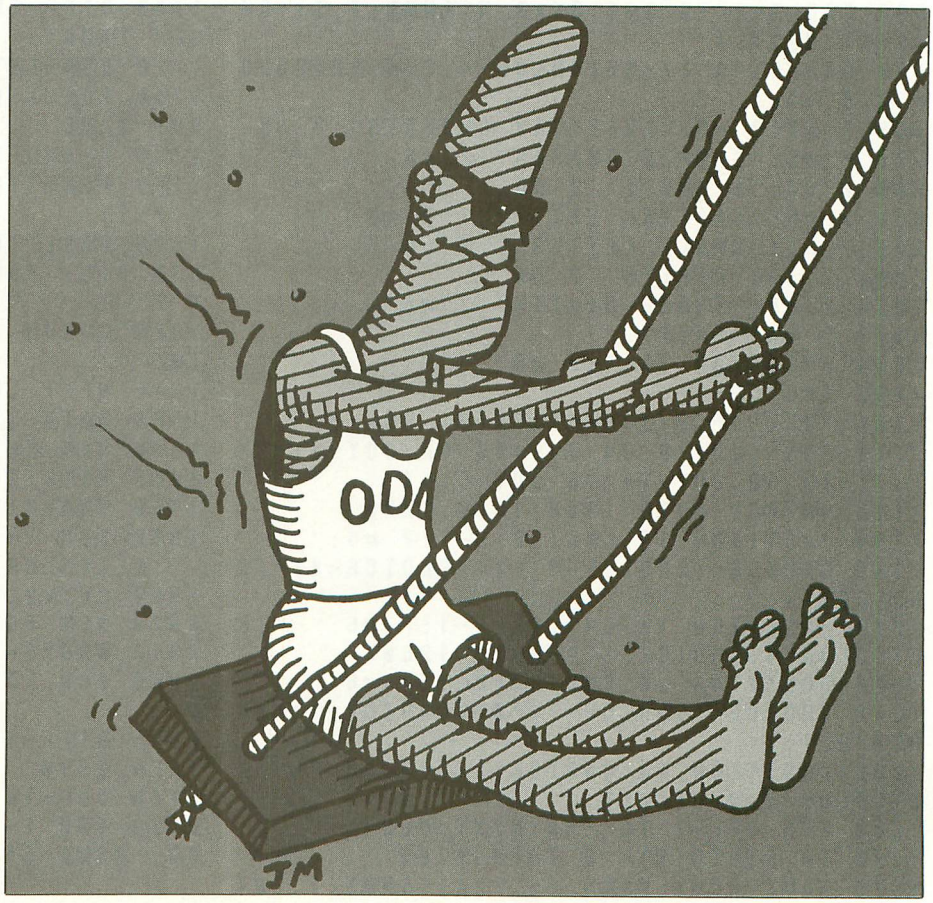
The next section of the code is also fairly straightforward. Lines 880–1080 present the menu in Graphics Mode 1 and set CHOICE to one. We then reset the Attract Mode Flag (POKE AF,0). This keeps the computer from automatically beginning the attract mode, since the keyboard is not used. Next, the joystick position is read. POS is then used to determine if the UP or DOWN motion on the menu is desired. POS is set to two if DOWN is desired or three if UP is indicated. The remainder of this section adjusts CHOICE to the appropriate value, changes the background color, moves the pointer, and checks to see if the fire button has been pressed.

The third section of the code copies the custom characters for the pushme-pullyou into the character set, and sets up the character-animation strings PMPY$ and ERASE$. Lines 1500–1860 replace some of the lines used last month with the proper code for levels 1, 2, and 3. This code is more interesting and requires some explanation.

Since Odd Man Out consists of ten problems, a FOR/NEXT loop is used to control problem setup. SAME selects the one object (out of eight possibilities) that is to be displayed three times in a particular problem. DIFF selects one of the remaining seven objects to be the odd object. ODD chooses the location (0 to 3) of the object chosen by DIFF. P points to the first character in the object to be displayed. The screen is printed and control is passed to the joystick-input routine at line 2500. Lines 1880–2480 are similar, except that they allow the standard Atari characters to be displayed as objects. The only modified characters are the answer box and the pushme-pullyou.

One line that may be confusing is 1940. To reduce the amount of work required, this line copies the lower-case letters into the first half of the character set at the locations of the capital letters (a=>A), rather than copying the pushme-pullyou, answer box, and other miscellaneous characters into the second half of the standard character set.

Lines 2500–2860 are both the heart of Odd Man Out and a place where Play Action becomes a serious consideration. POKE 752,1 turns off the cursor. Locations 656 and 657 hold the row and column, respectively, of the cursor in the Graphics window. The pushme-pullyou is displayed on the left-hand edge of the screen, and the joystick is checked for input. If the value of JOY is greater than seven, the joystick could not have been moved to the right, so control is passed to line 2660. If JOY is greater than 11,



this means that the joystick was not moved to the left, so control is passed to line 2800. If JOY<=11 or<=7, POS is updated and the character-animation sequence in lines 2700–2780 is activated.

The character-animation sequence displays the four frames required for the pushme-pullyou to take a step. By placing ERASE$ both before and after FRAME$, the left or right-hand edge of the pushme-pullyou that had been previously displayed will be erased. Lines 2740–2780 create the "clip-clop" sound effects of the walking character. The remainder of the section checks to see if the fire button has been pressed and if the pushme-pullyou is under one of the answer boxes. If it is, the next problem is displayed. Next month we'll add the right and wrong answer routines that need to be activated at this point.

PROGRAM ADDITIONS

It is now time to add this new section of code to the section presented in last month's column. Combining these two listings will provide you with the skeleton of Odd Man Out. However, there is still no indication in the program of which responses are correct or incorrect, or of an overall evaluation of a child's performance. These topics will be examined next month.

To combine the two listings, first type in Listing 1 and LIST it to disk or cassette tape. Then load last month's listing, ENTER the new listing, and save the resulting program to cassette or disk for future use.

```
10 REM ******************************
20 REM *                            *
30 REM *        ODD MAN OUT         *
40 REM *             BY             *
50 REM *   JOHN AND MARY HARRISON   *
60 REM *                            *
70 REM *             FOR            *
80 REM *       ANTIC MAGAZINE       *
90 REM *                            *
95 REM ******************************
140 DIM ANM$(104),NAME$(20),PMPY$(48),
FRAME$(12),ERASE$(12),DISP$(20)
180 AF=77
320 FRAMES=4:FRMSZE=10:POKE AF,0
560 FOR I=1 TO 104:READ X:ANM$(I,I)=CH
R$(X):NEXT I
880 GRAPHICS 17:SETCOLOR 4,2,8:SETCOLO
R 0,0,14
900 ? #6:? #6;"EVERYDAY OBJECTS":? #6
920 ? #6;"SIMPLE SHAPES":? #6
940 ? #6;"E AND [":? #6
960 ? #6;"CAPITAL LETTERS":? #6
980 ? #6;"SMALL LETTERS":? #6
1000 ? #6;"DIGITS":? #6
1020 ? #6;"THREE DIGITS":? #6
1040 ? #6;"EXIT"
1060 POSITION 18,1:? #6;"<"
1080 CHOICE=1
1100 POKE AF,0:JOY=STICK(0):POS=0
1120 POS=(JOY=9 OR JOY=13 OR JOY=5)+2*
(JOY=10 OR JOY=14 OR JOY=6)+1
1140 ON POS GOTO 1300,1160,1220
1160 POSITION 18,CHOICE*2-1:? #6;" "
1180 CHOICE=(CHOICE=8)*1+(CHOICE<8)*(C
HOICE+1)
1200 POSITION 18,CHOICE*2-1:? #6;"<":S
ETCOLOR 4,CHOICE*2,8:GOTO 1280
1220 POSITION 18,CHOICE*2-1:? #6;" "
1240 CHOICE=(CHOICE=1)*8+(CHOICE>1)*(C
HOICE-1)
1260 POSITION 18,CHOICE*2-1:? #6;"<":S
ETCOLOR 4,CHOICE*2,8
1280 FOR DEL=1 TO 100:NEXT DEL
1300 IF STRIG(0)<>0 THEN 1100
1320 RAM$=ROM$:RAM$(25,128)=ANM$(1,104
)
1340 RAM$(473,488)=BOX$
1380 PMPY$="#$%↓←←←&'()*%↓←←←+,-"
1400 PMPY$(21)="#$%↓←←←&'(#./↓←←←+,-"
1420 ERASE$=" ←↑ ←↑ "
1440 ON CHOICE GOTO 1480,1500,1520,196
0,1940,2240,2240,4140
1500 RAM$(145,208)=GEO$(1,64):RAM$(257
,448)=GEO$(65,256):GOTO 1540
1520 RAM$(145,208)=E$(1,64):RAM$(257,4
48)=E$(65,256)
1640 WRONG=0
1660 FOR LOOP=1 TO 10
1680 SAME=INT(8*RND(0))
1700 DIFF=INT(8*RND(0)):IF DIFF=SAME T
HEN 1700
1720 ODD=INT(4*RND(0))
1740 FOR I=0 TO 3:P=(I=ODD)*DIFF+(I<>0
DD)*SAME
1760 P=INDEX(P)+32
1780 POSITION I*4+3,5:? #6;CHR$(P);CHR
$(P+1)
1800 POSITION I*4+3,6:? #6;CHR$(P+2);C
HR$(P+3):POSITION I*4+3,8:? #6;"[":NEX
T I
1820 GOSUB 2500:REM JUMP TO INPUT ROUT
INE
1840 NEXT LOOP
1860 GOTO 3100
1880 REM SET UP FOR LEVELS IV, V
1940 RAM$(265,472)=RAM$(777,984)
1960 GRAPHICS 2:POKE 756,GRTOP
1980 FOR I=0 TO 3:SETCOLOR I,0,14:NEXT
 I:SETCOLOR 4,CHOICE*2,8
2000 SETCOLOR 2,CHOICE*2,8
2020 REM POKE 16,112:POKE 53774,112
2040 WRONG=0
2060 FOR LOOP=1 TO 10:SAME=INT(26*RND(
0))
2080 DIFF=INT(26*RND(0)):IF SAME=DIFF
THEN 2080
2100 ODD=INT(4*RND(0))
2120 FOR I=0 TO 3:P=(I=ODD)*DIFF+(I<>0
DD)*SAME+65
```

continued on page 24