Tetris for Applesoft BASIC

PROGRAM LISTING BY MARK STOCK, ANALYSIS BY APRIL AYRES-GRIFFITHS

What do Steve Wozniak and George H.W. Bush have in common? They've both been seriously into Tetris! But who can blame them? The object of the game (as if you didn't know) is to complete horizontal lines using falling shapes of various configurations. When you finish a line, it disappears, causing the rest of the blocks to fall down a line. However, if you stack up shapes to the point they overflow the top of the playfield: Game Over. When you finish a certain number of lines, the level ends... and in the next, the shapes fall faster, and you need to complete more lines! The insanity never ends.

In an era where games were becoming increasingly more complex, the simplicity of Tetris was seen as a breath of fresh air. Tetris would inspire a number of other "falling block puzzle games" such as Sega's colour-matching Columns, and the three-dimensional Welltris. But Tetris would always remain king (tsar?) of the

arcade puzzle game world, with sequels, clones and variations being released for virtually every console, computer, and operating system worldwide.

A little history

Alexey Pajitnov's Tetris – early unauthorised versions of which, such as Spectrum Holobyte's rendition described above, began appearing on home computers in 1987 – spawned a whole new generation of shape-based puzzle games.

Holobyte's version took next year's CES by a storm, and garnered the attention of the Soviet government, who held the rights to the game. They sold the arcade rights to Atari and the console rights to Nintendo.

Nintendo released Tetris on both its Nintendo Entertainment System and on the Game Boy, the latter as a pack-in with the portable console. The inclusion of Tetris arguably made the Game Boy a success –the game was perfect for smaller screen sizes, and was very addictive, spawning a whole generation of Tetris 'junkies'.

Nintendo's version of Tetris for the NES was criticised for not having a two-player mode; however, Atari Games, perhaps mistakenly believing their arcade licensing gave them the right to release a console version, came out with their own Nintendo Tetris game through their Tengen subsidiary (created after the consumer rights to the Atari name were sold to Jack Tramiel, see Point & Click) which featured head-to-head play.

Nintendo sued, and Tengen was eventually forced to withdraw its cartridge from sale, after selling around 100,000 copies. They are collectibles.

About the listing

This Tetris listing for the Apple II has a number of interesting features that suggest the author has taken great pains to make the listing as efficient as possible.

Constants vs. Variables

For one thing, rather than using constant values such as "1" or "2" in arithmetic expressions, they have opted instead to use variables. This is a lot more efficient in Applesoft, as using constants requires the parser to convert a string in the listing into a number each time that it is encountered, rather than using a variable

which stores the values in a format that is ready to use. You can also see this trick being used with memory addresses, for example the keyboard address is stored in a variable called “KB”.

Finding the right line for all occasions

Additionally, subroutines which need to be called frequently (for example the block drawing routines), are towards the top of the program, and less frequently called subroutines (for example game over) are towards the end of the listing. Although this might at first glance seem rather arbitrary, there is a reason for structuring the program this way.

Applesoft programs are stored in memory as a series of statements with a known starting address (typically 2049 in memory). In order to locate a given line of the program (for example if using a GOTO or a GOSUB) then we need to find the right line.

In order to do this, the Applesoft interpreter will start at the beginning, and check each line number. In memory, each line begins with a two byte line number, followed by two bytes which point to the next line numbers address in memory. So to find line 1000, we need to look at each line and if it isn’t the correct one, then move to the address of the next line and check that.

This seems like a lot of work, right? The more of the program it has to search through for the right line, the longer each GOTO or GOSUB will take to perform. So we can speed things up a lot by putting things that need to happen frequently, (and quickly) at the start of the program. This makes the game faster as it spends more time doing what we want it to do, and less time trying to find the next line.

Why use code when data will do

The final thing that is interesting about this listing, is that rather than trying to define the various shapes in the game using code, which can become inefficient rather quickly, the author has opted to define the shapes themselves as data statements, which are read into variables.

This makes the program easier to amend in future, should they wish to change the shapes, but also means much more simplified logic can be used in the game.

Where possible, let data define the game behaviour rather than hard-coded logic. This is especially true when your program involves multiple different things that follow the exact same rules.

The listing (finally!)

This Applesoft BASIC program can be typed into a real Apple II (the preferred experience, although the Apple II's keyboard could quickly become tiresome), or an Apple II emulator such as *microM8* (Mac / Windows) or *AppleWin* (Windows)

In *microM8* you can choose 'A' from the boot menu to go into Applesoft BASIC. If you use *AppleWin* you will need to boot from a DOS disk image.

Type in the numbered lines from the left-hand colum of each page. The notes in the right-hand column describe what the numbered lines do. **Don't forget to save often!**

Aww typing... what is this, 1985? Yes! Typing in a program is not without value. (Click on each page for full-sized image)

TETRIS

Program Listing

by Mark Stock

This Applesoft BASIC program can be typed into a real Apple II (the preferred experience, although the Apple II's keyboard could quickly become tiresome), or an Apple II emulator such as *microM8* (Mac / Windows) or *AppleWin* (Windows) See download links to the right.

In *microM8* you can choose 'A' from the boot menu to go into Applesoft BASIC. If you use *AppleWin* you will need to boot from a DOS disk image.

Type in the numbered lines from the left-hand colum of each page. The notes in the right-hand column describe what the numbered lines do. **Don't forget to save often!**

```
10  GOSUB 1000

100  W = W + 1 : IF W > LV THEN W = 0 : GOSUB 350

110  K = PEEK( KB ) : IF K >= H THEN POKE KC , H : K = K - H
: GOSUB 300

190  GOTO 100

200  PY = PY * A2 : VLIN PY , PY + A1 AT PX : RETURN

225  PY = PY * A2 : HLIN X1 , X2 AT PY : HLIN X1 , X2 AT PY
+ A1 : RETURN

300  ON E ( K ) GOTO 30000 , 330 , 340 , 350 , 360 , 30100

310  RETURN
```

Apple II emulators:

microM8:	**http://microm8.com**
AppleWin:	**https://github.com/AppleWin/AppleWin/releases**
DOS disk image for AppleWin:	**https://archive.org/download/Apple_DOS_3.3_Master/Apple_DOS_3.3_Master.dsk**

This Tetris clone uses the Apple II's low-resolution graphics mode. In this mode, the Apple II can display a grid of up to 40x48 pixels, each from a palette of 16 colours. You enter the "low-res" mode using the BASIC command GR (for 40x40 pixels with a 4 line text "window" at the bottom) or GR2 for full screen graphics. The TEXT command returns to text mode.

The BASIC command VLIN draws a vertical line (PY) from one horizontal (PX) pixel to another, and HLIN does similarly from one vertical (PY) pixel to another. PLOT "plots" a single pixel at X,Y.

Although it looks like a variable assignment, COLOR= is actually a command! It sets the colour used by PLOT, VLIN or HLIN. Other notable commands in this listing include DEF FN (which defines a mathematical function that can be used over and over), ON X (jumps to one of a series of specified line numbers based on the value of X) and PEEK (which returns the value stored in memory at the specified location). For more information on Applesoft BASIC, see **http://www.calormen.com/jsbasic/reference.html**

Line 10: Call subroutine at line 1000 to initialise game.

Line 100: Increment W by 1, if it is greater than LV move the shape down using subroutine at line 350. LV acts as a delay. The lower LV is, the faster the shapes will fall down the screen.

Line 110: Read keyboard character and based on keypress handle action using subroutine at line 300.

Line 200: Draw square block (1x2) at PX, PY. This is used to draw the tetris shapes.

Line 225: Draw 2 block thick horizontal line between X1 and X2 at position PY.

Line 300: Based on the key pressed, jump to another line. Remember that keys were defined at line 1130.

```
330  X = X - 1 : GOTO 400

340  X = X + 1 : GOTO 400

350  DN = 1 : Y = Y + 1 : GOSUB 400 : DN = 0 : RETURN

360  S = S + 1 : IF S / 4 = INT( S / 4 ) THEN S = S - 4

400  GOSUB 500

410  GOSUB 800 : IF F = 0 THEN X = XX : Y = YY : S = SS :
GOSUB 420 : IF DN THEN GOSUB 900

420  COLOR= CF : FOR PP = 1 TO 4 : PX = X + X ( S , PP ) :
PY = Y + Y ( S , PP ) : GOSUB 200 : NEXT PP : XX = X : YY =
Y : SS = S : D = 0 : RETURN

500  IF DD THEN RETURN

510  COLOR= CB : FOR PP = 1 TO 4 : PX = XX + X ( SS , PP )
: PY = YY + Y ( SS , PP ) : GOSUB 200 : NEXT PP : DD = 0 :
RETURN

800  F = 1 : FOR PP = 1 TO 4 : PY = Y + Y ( S , PP ) : ON (
FN PC ( X + X ( S , PP ) ) > 0 ) GOTO 805 : NEXT PP : RETURN

805  F = 0 : RETURN

850  F = 1 : RETURN

900  P = 10 : GOSUB 30300

905  RN = 0 : Y = YM

910  X = XL

920  PY = Y : IF FN PC ( X ) = CB THEN 950

930  X = X + 1 : IF X <= XR THEN 920

940  R ( RN ) = Y : RN = RN + 1

950  Y = Y - 1 : IF Y >= 0 THEN 910

960  IF RN THEN GOSUB 30400

970  Y = 0
```

Line 310: Falls through here if E(K) equals zero.

Line 330: Move shape 1 space left.

Line 340: Move shape 1 space right.

Line 350: Move shape down 1 unit.

Line 360: Rotate shape.

Line 400: Start of shape drawing routine. Calls to 500 to clear previous shape.

Line 410: Call subroutine to see if space is clear. If not we will draw the shape in its previous position.

Line 420: Draw current shape position.

Line 510: Undraw previous shape (draw over previous position and rotation with black pixels). Note, we use XX, YY and SS here which are the previous shape position.

Line 800: Check to see if any space where we are going to draw the shape already has a block there. If so, go to line 805.

Line 805: Set F to zero, then return

Line 850: Set F to one then return

Line 900: Add 10 points to the score.

Line 920: Check if current position is empty (no colored blocks). If true, go to line 950, otherwise continue.

Line 930: Move shape right. If ok, then go to 920 to check if the Y position is empty as well.

Line 950: Move shape up one line. If still on screen then go to line 910.

Line 970: Set shape Y position to zero

```
980  X = INT( ( XR - XL ) / 2 ) + XL

985  S = INT( RND( 1 ) * NS ) : CF = C ( S ) : S = S * 4

990  GOSUB 800 : IF F THEN RETURN

995  GOTO 31000

1000  DIM E ( 127 ) , X ( 27 , 4 ) , Y ( 27 , 4 ) , R ( 40 )

1010  TEXT : HOME : GR

1011  PRINT "WELCOME..."

1014  LM = 10

1015  XM = 10 : YM = 15

1016  XL = INT( ( 40 - XM ) / 2 )

1017  XR = XL + XM - 1

1021  A1 = 1

1022  A2 = 2

1030  DEF FN PC ( X ) = SCRN( X , PY * A2 )

1040  CB = 0

1050  XX = 20 : YY = 0 : SS = 0

1100  KB = -16384

1110  KC = -16368

1120  H = 128

1129  REM KEYBOARD ACTIONS

1130  REM QUIT
```

Line 980: Set shape X position to half way across play area.

Line 985: Choose a random shape from the shapes available.

Line 990: Check if the shape can be drawn there without colliding with other blocks. If so, return.

Line 995: If we are here, do the "GAME OVER" routine at 31000 as the play area is filled to the top.

Line 1000: Dimension and setup variables. E is used to map keypresses to functions. X and Y store information about the blocks that make up each of the game shapes.

Line 1010: Clear the screen and enter LORES graphics mode.

Line 1011: Display a welcome message.

Line 1015: Define the size of the play area in blocks.

Line 1016: Calculate the left edge of the play area.

Line 1017: Calculate the right edge of the play area.

Line 1021: A1 is used as a constant for speed.

Line 1022: A2 is used as a constant for speed. Each horizonatal line or block is 2 pixels high.

Line 1030: Define a function to read the pixel color at position X, PY

Line 1040: Define background color (0) which is black. This represents empty space in the play area.

Line 1050: XX is Previous shape X, YY is Previous shape Y and SS is previous shape pointer.

Line 1100: KB is defined as a constant used for the address of the Apple II keyboard buffer.

Line 1110: KC is defined as a constant used for the address

```
1131  E ( ASC( "Q" ) ) = 1
1132  E ( ASC( "Q" ) - 64 ) = 1
1140  REM MOVE LEFT
1141  E ( 8 ) = 2
1142  E ( ASC( "," ) ) = 2
1150  REM MOVE RIGHT
1151  E ( 21 ) = 3
1152  E ( ASC( "." ) ) = 3
1160  REM MOVE DOWN
1161  E ( 32 ) = 4
1162  E ( ASC( "Z" ) ) = 4
1170  REM ROTATE
1171  E ( ASC( "R" ) ) = 5
1172  E ( 13 ) = 5
1173  E ( ASC( "A" ) ) = 5
1179  REM PAUSE GAME
1180  E ( ASC( "P" ) ) = 6
1181  E ( ASC( "P" ) - 64 ) = 6
1185  GOSUB 2000
1186  GOSUB 1300
1190  PRINT "PRESS ANY KEY TO START..."
1191  PRINT
1192  PRINT "PRESS Q TO QUIT."
1193  GOTO 31020
1299  REM DRAW THE GAME
```

of the Apple II key strobe.

Line 1120: H is defined as 128, it us used to check for new keypresses at address KB. If a new press, it will be bigger than 128.

Lines 1130 - 1181: Define keymappings between the ASCII key-code and a function number (1-6). These are used to handle actions during the game.

Line 1185: Call subroutine to load share data from DATA statements.

Line 1186: Call subroutine to draw play area outside.

Lines 1190 - 1193: Display instructions for starting and quitting the game, and jump to keyboard reading subroutine.


```
1300  COLOR= 4 : FOR I = 0 TO 19 : X1 = 0 : X2 = 39 : PY = I
: GOSUB 225 : NEXT
1320  COLOR= CB : FOR I = 0 TO YM : X1 = XL : X2 = XR : PY =
I : GOSUB 225 : NEXT
1350  RETURN
1400  DATA 1
1401  DATA 0,0,1,0,0,1,1,1
1402  DATA 0,0,1,0,0,1,1,1
1403  DATA 0,0,1,0,0,1,1,1
1404  DATA 0,0,1,0,0,1,1,1
1410  DATA 2
1411  DATA 0,1,1,1,2,1,3,1
1412  DATA 1,0,1,1,1,2,1,3
1413  DATA 0,1,1,1,2,1,3,1
1414  DATA 1,0,1,1,1,2,1,3
1420  DATA 12
1421  DATA 1,1,0,1,1,0,2,1
1422  DATA 1,1,0,1,1,0,1,2
1423  DATA 1,1,0,1,2,1,1,2
1424  DATA 1,1,1,0,2,1,1,2
1430  DATA 13
```

Lines 1300 - 1350: Draw the game screen. First fill with green, then draw the play area in black.

Lines 1400 - 1990: This holds the shape data for each type of shape.

First is the shape color, then follows 4 versions of the shape, each rotated 90 degrees. Each contains 4 (X, Y) pairs.

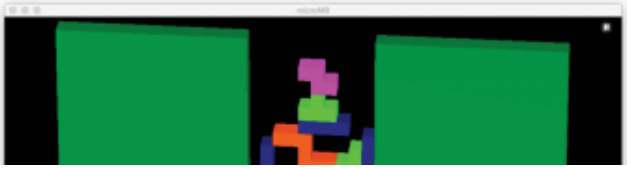
Line 1410: Shape 2 color

Line 1411: Shape 2 first rotations (X, Y pairs)

Line 1412: Shape 2 second rotation (X, Y pairs)

Line 1413: Shape 2 third rotation (X, Y pairs)

Line 1414: Shape 2 forth rotation (X, Y pairs)



```
1431  DATA 1,1,0,1,2,1,0,2
1432  DATA 1,1,1,0,1,2,2,2
1433  DATA 1,1,0,1,2,1,2,0
1434  DATA 1,1,1,0,1,2,0,0
1440  DATA 9
1441  DATA 1,1,0,1,2,1,2,2
1442  DATA 1,1,1,0,1,2,2,0
1443  DATA 1,1,0,1,2,1,0,0
1444  DATA 1,1,1,0,1,2,0,2
1450  DATA 3
1451  DATA 1,1,1,0,0,0,2,1
1452  DATA 1,1,1,0,0,1,0,2
1453  DATA 1,1,1,0,0,0,2,1
1454  DATA 1,1,1,0,0,1,0,2
1460  DATA 6
1461  DATA 1,1,0,1,1,0,2,0
1462  DATA 1,1,0,1,0,0,1,2
1463  DATA 1,1,0,1,1,0,2,0
1464  DATA 1,1,0,1,0,0,1,2
1990  DATA -1
2000  X = 0 : Y = 0
2010  NS = 0
2020  READ C : IF C <> -1 THEN C ( NS ) = C : FOR J = 0 TO 3
: FOR I = 1 TO 4 : READ X ( NS * 4 + J , I ) : READ Y ( NS *
4 + J , I ) : NEXT I : NEXT J : NS = NS + 1 : GOTO 2020
2030  RETURN
21210  P = 1 : RETURN
```

Line 1990: -1 signals end of data.

Line 2000: Reset X and Y to zero.

Line 2010: NS (Number of shapes) set to zero.

Line 2020: Read in shape data using the format described above (see line 1410).

In microM8 you can play your new Tetris game in 3D! Once the game starts, pause microM8 by pressing Control-Shift-Space. Then you can move the "camera" by using the following key combinations:

- **Hold Shift-Control-Alt/Option and press the arrow keys to "orbit" around the model.**
- **Hold Control-Alt/Option and press the arrow keys to move the model.**
- **Hold Shift-Alt/Option and press the up or down arrow keys to zoom in and out.**
- **Hold Shift-Alt/Option and press the left or right arrow keys to rotate the model**

This trick works with any game you play in microM8 (although in high resolution programs you may need to turn on 3D rendering by pressing Shift-Control-G, releasing it and quickly pressing the number 2.)

You can download microM8 from *http://microM8.com*


```
30000  TEXT : HOME : END
30100  HOME
30110  PRINT "GAME PAUSED.  PRESS P TO CONTINUE..."
30120  P = 1
30130  K = PEEK( KB ) : IF K >= H THEN POKE KC , H : K = K -
H : GOSUB 30200
30140  IF P THEN 30130
30150  HOME
30160  PRINT "SCORE " ; SC ; TAB( 21 ) ; "LEVEL " ; LM - LV
+ 1
30170  RETURN
30200  ON E ( K ) GOTO 30000 , 30210 , 30210 , 30210 , 30210
, 30220
30210  RETURN
30220  P = 0
30230  RETURN
30300  SC = SC + P
30310  VTAB 21 : HTAB 7
30320  PRINT SC ;
30330  RETURN
30400  RN = RN - 1
```

Line 30000: Clear the screen into text mode and end the program.

Line 30100: Clear screen and display pause message.

Line 30130: Read the keyboard. If a key is pressed call subroutine at 30200.

Lines 30150 - 30160: Refresh the score on the screen.

Line 30200: Based on keypress and mapping in array E, call an appropriate subroutine.

Lines 30300 - 30330: Update the score by adding P to it and redisplay it on the screen.

Lines 30400 - 30440: Flash horizontal lines on the screen.

```
30410  FOR C = 0 TO 32

30415  COLOR= C

30420  FOR I = 0 TO RN : X1 = XL : X2 = XR : PY = R ( I ) :
GOSUB 225 : NEXT I

30430  FOR I = 0 TO 2 : NEXT I

30440  NEXT C

30450  FOR I = 0 TO RN                                    Line 30450-30490: Remove completed lines from the screen.

30460  Y = R ( I ) + I

30470  YP = Y - 1 : FOR X = XL TO XR : PY = YP : COLOR= FN
PC ( X ) : PX = X : PY = Y : GOSUB 200 : NEXT X : Y = Y - 1
: IF Y > 0 THEN 30470

30480  P = 100 : GOSUB 30300

30490  NEXT I

30495  RETURN

31000  VTAB 22 : PRINT                                    Lines 31000-31010: Display "GAME OVER" message.

31010  PRINT "              GAME OVER"

31020  P = 1

31030  K = PEEK( KB ) : IF K >= H THEN POKE KC , H : K = K -   Line 31030: Read keyboard. If a key has been pressed call
H : GOSUB 31200                                           subroutine at line 31200 to handle.

31040  IF P THEN 31030

31050  D = 1

31060  SC = 0 : LV = LM                                   Line 31060: Reset score and level (speed).

31070  GOSUB 30150

31080  GOSUB 1300                                         Line 31080: Call subroutine to redraw the game screen.

31090  GOTO 905                                           Line 31090: Continue game.

31200  ON E ( K ) GOTO 30000                              Line 31200: If a valid key has been pressed, jump to line
                                                          30000
31210  P = 0 : RETURN

32000 REM END OF LISTING                                  Line 32000: This is the end! Type RUN and cross your fin-
                                                          gers...
```