Apple Max Code Description

Basic 10-Liner for the NOMAM 2015 Competition

by Cliff Hatch

Recursive Tree

Lines 5 to 8 comprise a recursive subroutine which draws the apple tree.

On the first call (GOSUB 5 in line 3) it draws the first segment of the tree, a vertical trunk with a single leaf at the end. It then calculates the angle and length of two branches and calls itself to draw these – and so on, until the required depth of calls is reached.

Turbo Basic provides no support for recursion apart from a call stack. Hence the program has to stack the five variables defining each segment of the tree itself:

- x - coordinate of start of segment.
- y - coordinate of start of segment.
- a - angle (degrees).
- l – length (pixel widths).
- j – branch number (1 or 2).

The stack is the array S(45). It contains groups of the above variables in S(0-4), S(5-9), S(10-14), etc.. The variables are accessed using pointers (or subscripts) X, Y, A, L and J.

The recursive call is the GOSUB 5 instruction in line 8. Prior to the call, the appropriate data is stacked, and the stack pointers are increased by 5. On return, the pointers are decreased by 5.

Leaf Area

The tree was designed with variegated leaves primarily to make it look better, but also to make it easier for the player to see excessively overlapped leaves.

When the program calculates leaf area, it counts the peripheral dark green pixels only, ignoring the light green (almost yellow) ones. This helps to keep the code simple. It also penalises large leaves because they are not credited with their full area. The penalty is further trimmed by an exponent (N) in the score calculation.

The penalty is intended to reflect the greater structural demands imposed by large leaves in nature.

Variables

Variable	Description
A	Angle of branch (0 degrees is up, -90 left and +90 right).
C	The radius of the sun (pixel widths).
D	Gene controlling the amount of branching and the number of leaves. Used in the code as a call depth indicator, decrementing to zero at maximum depth. D is also the number of straight segments connecting ground level to the outermost leaves.
E	Copy of D for high score data.
F	Gene controlling the size reduction of successive branch segments (%).
G	Copy of F for high score data (%).
H	X coordinate for drawing and scanning leaves.
I	Y coordinate for drawing and scanning leaves.
J	Stack pointer to branch number (1 or 2)
K	Copy of R for high score data.
L	Stack pointer to branch segment length.
M	Genes OK flag (One or more genes out of range = 0, all OK = 1).
N	Exponent in the score calculation, expressing greater penalties for larger leaves.
O	High score (apples).
P	Retrieved colour of pixel, used during scan.
Q	sunin, amount of energy absorbed by the tree, determined by the exposed leaf area.
R	Leaf radius (pixel widths).
S(45)	Stack containing groups of 5 variables per branch segment (referenced by pointers X, Y, A, L and J). 45 array elements are required to support recursive calls to depth 8. That is 40 + 5 extra to receive data for a 9^{th} call (although the 9^{th} call never occurs, some variables are stacked in advance).
T	Leaf fill counter.
U	leafg, energy required to grow the leaves, determined by the total leaf area.
V	brang, energy required to grow the branches, determined by the total length of all branch segments (pixel widths).
W	Number of leaves.
X	Stack pointer to x coordinate.
Y	Stack pointer to y coordinate.
Z	Score. The number of apples the tree is capable of producing per season. If negative the genome is not viable and the tree will die without producing any apples.

Code Comments

0 DIM S(45),Q$(1):X=0:Y=1:A=2:L=3:J=4:S(X)=80:S(Y)=80:S(A)=0:S(L)=30:C=10:O=-9999:N=2.7

- Dimension the stack, S(45).
- Initialise stack pointers X, Y, A, L and J.
- Set parameters on the first level of the stack to define the first segment of the tree.
- Initialise high score, O.
- Initialise N, an exponent used in the score calculation, expressing greater penalties for larger leaves.

1 V=0:W=0:M=1:GRAPHICS 7:SETCOLOR 0,1,4:SETCOLOR 1,12,14:SETCOLOR 2,11,4:COLOR 2:CIRCLE C,C,C:PAINT C,C:? "ÇÅÎÅ ÒÁÎÇÅÓº Ä ±-¸¬ Æ µ°-¶¸¬ Ò ±-´ "

- Initialise the total branch length V and the leaf counter W to zero.
- Initialise the genes OK flag to 1.
- Set graphics mode and colours.
- Draw the sun.
- Print gene ranges for reference.

2 IF O>-9999:? "Hi(";E;",";G;",";K;")";O,;:ENDIF :INPUT "D,F,R? ",D,F,R:IF D<1 OR D>8 OR F<50 OR F>68 OR R<1 OR R>4 THEN M=0

- Print high score if it has been set.
- Input genes D, F and R. Check if they are within range and set flag M accordingly (0 = not all in range, 1= all in range).

3 IF M=0:? "ý}";:GOTO 1:ELSE :GOSUB 5:GOSUB 9:ENDIF :U=W*8*R^N/32 DIV 1:V=V DIV 1:? "sunin=";Q;", leafg=";U;", brang=";V

- If any genes are out of range go to line 1 to input them again.
- GOSUB 5 to draw the tree.
- GOSUB 9 to scan the tree and measure the exposed dark green leaf area.
- Calculate the energy required to grow the leaves and truncate to an integer. The dark green area of each leaf is 8 * R. The constants N and 32 were set by trial and error to allow the creation of a reasonably diverse set of viable trees. The exponent N trims the penalty for large leaves.
    - U = W * 8 * R^N / 32 DIV 1
- Truncate the exposed green leaf area V to an integer.
    - V = V DIV 1
- Print the incoming energy (sunin, Q), energy consumed in leaf growth (leafg, U) and energy consumed in branch growth (brang, V).

4 Z=Q-U-V:? "ÓÃÏÒÅº ";Z;:IF Z>=0:? " apples";:ELSE :? ", failed";:ENDIF :IF Z>O:E=D:G=F:K=R:O=Z:ENDIF :? ,"Replay";:INPUT Q$:CLS :GOTO 1

- Calculate the score:
    - Z = Q – U - V
- Print “apples”, or “failed” if the score is negative.

```
5 IF D>0:COLOR 1:PLOT S(X),S(Y):S(X+5)=S(X)+S(L)*SIN(S(A)):S(Y+5)=S(Y)-
S(L)*COS(S(A)):DRAWTO S(X+5),S(Y+5):V=V+2*S(L)
```

- Start of subroutine to draw the tree.
- If call depth D=0 the tree is complete, just return.
- Plot the start point of the segment and draw to its end.
- Set the start coordinates for the next segment on the stack.

```
6   W=W+1:H=S(X+5)+((R+1)*SIN(S(A))):I=S(Y+5)-((R+1)*COS(S(A))):COLOR 2:PLOT
H,I:T=1:WHILE T<R:CIRCLE H,I,T:T=T+1:WEND :COLOR 3
```

- Increment the leaf counter.
- Draw the light green centre of the leaf at the end of the segment.

```
7   CIRCLE H,I,R:COLOR 1:S(J)=0:WHILE S(J)<2:S(A+5)=S(A)-
45+(90*S(J)):S(L+5)=S(L)*F/100:X=X+5:Y=Y+5:A=A+5:L=L+5:J=J+5
```

- Draw the dark green periphery of the leaf.
- Start loop to draw the next two segments.
- Set remaining parameters defining the next segment on the stack
- Increase the stack pointers by 5.

```
8     D=D-1:GOSUB 5:D=D+1:X=X-5:Y=Y-5:A=A-5:L=L-5:J=J-5:S(J)=S(J)+1:WEND :ENDIF
:RETURN
```

- Decrement the call depth.
- Recursive call to draw remaining segments.
- Increment the call depth.
- Decrease the stack pointers by 5.
- End of subroutine to draw the tree.

```
9 ? "Scanning";:Q=0:FOR I=0 TO 76:IF I MOD 8=0:? ".";:ENDIF :FOR H=81 TO 140:LOCATE
H,I,P:IF P=3:Q=Q+2:ENDIF :NEXT H:NEXT I:POKE 657,2:RETURN
```

- Scan the tree counting the number of dark green pixels.
- Only need to scan the right hand side because the tree is symmetrical. The centre line down the trunk is omitted from the scan, which introduces some insignificant errors into the pixel count.