

APPLE-1 LOGO V2.6

Turtle interpreter for the P-LAB TMS9918 Graphic Card on the Apple-1

(C) 2026 VERHILLE Arnaud

Table of Contents

1. What is APPLE-1 LOGO?
 2. Setup and First REPL
 3. The Turtle
 4. Colour
 5. Repetition and Control Flow
 6. Variables and Arithmetic
 7. Procedures
 8. Sprites
 9. Bitmap Text
 10. Built-in Demos
 11. Tutorials by Example
 12. Reference Card
 13. Limitations
 14. Internals (for the curious)
 15. Hardware and Memory Map
-

1. What is APPLE-1 LOGO?

APPLE-1 LOGO V2.6 is a complete LOGO turtle interpreter that runs on a real Apple-1 (or POM1 emulator) equipped with the **P-LAB TMS9918 Graphic Card**. It draws on the TMS9918's 256×192 bitmap (Mode 2) using the chip's hardware sprites for animated turtles.

It is a *real* LOGO with:

- A turtle that responds to `FORWARD` , `RIGHT` , `LEFT` , `HOME` , etc.
- User-defined procedures with parameters and recursion.
- Variables and a single level of arithmetic in argument position.
- `IF` / `IFELSE` / `STOP` , `REPEAT` , `REPEAT FOREVER` (ESC-abortable).
- A unified colour command, hardware sprite turtles, bitmap text.
- A fullscreen line editor for procedures (`EDIT`) and a proc dumper (`LIST`).
- Two slideshows (`DEM0` , `DEM2`) for showing off.

Source: `TMS_Logo_16k.asm` , linked at `$4000` and shipped on the CodeTank daughterboard (upper bank). Flip the CodeTank jumper to **Upper** and start the interpreter with `4000R` from Wozmon.

2. Setup and First REPL

In POM1

1. Launch POM1 and pick **Preset 2** (P-LAB Apple-1 with TMS9918 + CodeTank daughterboard).
2. Flip the CodeTank jumper to **Upper** in the Hardware menu, then in Wozmon (the `\` prompt) type: `4000R`
3. The TMS9918 screen opens, the turtle (a small triangle) appears at centre, and the prompt waits: `APPLE-1 LOGO FOR TMS9918 - TYPE HELP (C) 2026 VERHILLE ARNAUD ?`

On a real Apple-1

You need an Apple-1, the P-LAB TMS9918 card, and the CodeTank daughterboard. Flash `Codetank_GAME1.rom` into the 28C256, set the jumper to Upper, then type `4000R` from Wozmon.

Saying hello

Try:

```
PRINT "HELLO
```

The console echoes `HELLO`. The `?` prompt comes back.

Try a turtle move:

```
FD 60  
RIGHT 90  
FD 60
```

Two perpendicular trails of 60 pixels each.

Always available:

```
HELP
```

prints the table of contents. `HELP 1` through `HELP 8` page through the topical sections (turtle, console, variables, procs, dynamic turtle, demos, errors).

`BYE` returns to Wozmon.

3. The Turtle

The turtle has a position `(x, y)` in pixel coordinates ($0..255 \times 0..191$, origin top-left), a heading in degrees (0 = north, 90 = east), and a pen state (down / up).

Movement

Command	Aliases	What it does
FD <i>n</i>	FORWARD <i>n</i>	move <i>n</i> pixels in the current heading; if pen down, draw a trail
BK <i>n</i>	BACK <i>n</i>	move <i>n</i> pixels backward
TR <i>n</i>	RIGHT <i>n</i>	turn right (clockwise) by <i>n</i> degrees
TL <i>n</i>	LEFT <i>n</i>	turn left by <i>n</i> degrees
SETXY <i>x y</i>		jump to (<i>x</i> , <i>y</i>)
SETH <i>n</i>		set heading to <i>n</i> degrees (mod 360)
HOME		jump to centre (128, 96), heading 0, pen state preserved

n is unsigned 0..65535. Negatives are not parsed; use BK instead of FD -*n*.

Pen

Command	Aliases	What it does
PD	PENDOWN	turtle leaves a trail
PU	PENUP	turtle moves silently

HOME preserves pen state — a PU + HOME keeps the pen up so the next moves don't draw a line back to the origin.

Screen

Command	Aliases	What it does
CS	CLEARSCREEN	clear the bitmap (pixels only, not the colour table); HOME is implicit

A first picture

```
CS
FD 50
RIGHT 90
FD 50
RIGHT 90
FD 50
RIGHT 90
FD 50
```

A 50-pixel square, drawn from the centre.

4. Colour: the single SETPC command

V2.6 has exactly one colour command:

```
SETPC n          (n = 0..15)
```

It drives **every** colourised surface simultaneously:

- the turtle's **trail**;
- the **bitmap arrow** (the default ARROW shape) — its outline cells are repainted live;
- the **sprite-0** turtle (any other SETSHAPE value) — the sprite attribute is re-emitted with the new colour;
- **bitmap text** drawn by LABEL and the visual editor (LIST / EDIT).

The TMS9918 palette (Mode 2):

n	Colour	n	Colour
0	transparent	8	medium red
1	black	9	light red
2	medium green	10	dark yellow
3	light green	11	light yellow
4	dark blue	12	dark green
5	light blue	13	magenta
6	dark red	14	grey
7	cyan	15	white (<i>default</i>)

SAY (the speech bubble, see §9) is the one exception — it forces the bubble outline and glyph cells to white for readability, then restores your pen_color so subsequent commands keep the user-chosen tint.

Examples

```
SETPC 11          ; yellow
REPEAT 5 [FD 80 RIGHT 144]      ; 5-point yellow star
```

```
SETPC 9           ; red
SETSHAPE "ANGRY ; sprite turns red on the fly
```

The old V2.5 SETTC / SETSC commands are gone.

5. Repetition and Control Flow

REPEAT

```
REPEAT N [ ... ]
```

Run the bracketed block N times. Blocks may contain any commands, including more REPEAT s and proc calls.

```
REPEAT 36 [ FD 5 RIGHT 10 ] ; circle
```

```
REPEAT 12 [ REPEAT 6 [ FD 25 RIGHT 60 ] RIGHT 30 ] ; rosette
```

REPEAT FOREVER

```
REPEAT FOREVER [ ... ]
```

Loops until you press **ESC** or **Ctrl-G**. The interpreter aborts cleanly: any procs you defined are still there.

IF / IFELSE

```
IF a OP b [ ... ]
IFELSE a OP b [ yes-block ] [ no-block ]
```

OP is one of `<`, `>`, `=`, `<=`, `>=`, `<>` (not equal). `a` and `b` can be literal numbers or `:VARNAME` references.

```
IF :SIZE > 100 [ STOP ] ; common spiral termination
IFELSE :N = 0 [ PRINT "ZERO ] [ PRINT "NONZERO ]
```

STOP

STOP exits the current procedure immediately. Inside REPEAT, the loop unwinds and STOP propagates to the enclosing proc.

Quitting

```
BYE
```

Returns to Wozmon (if you launched LOGO from there).

6. Variables and Arithmetic

MAKE

```
MAKE NAME N
```

Create or update a variable named `NAME` with value `N`. `NAME` is up to **6 characters** (A-Z, 0-9). Leading underscores are not allowed.

Reading

`:NAME` evaluates to the variable's value.

```
MAKE SIZE 50
FD :SIZE
```

Arithmetic in argument position

Each argument supports a **single** operation:

```
FD :size + 2
TR 360 / :n
SPIRAL :s - 1 :a
```

Allowed operators: `+` `-` `*` `/`. Multiplication and division are 16-bit unsigned. There is no nesting: `:a + :b + 1` is **not** parsed as a chain — write it as a temporary `MAKE`.

RANDOM

```
RANDOM N
```

Returns a number in `0..N-1`. Backed by a 16-bit LFSR seeded at boot.

```
REPEAT 80 [ FD RANDOM 20 RIGHT RANDOM 90 ]
```

Limits

- 12 named variables maximum (the table is small to leave room for procs).
- 6-character name cap — `THING` and `THING1` stay distinct, but `LONGNAME` and `LONGNAMA` collide on the first 6 chars.

7. Procedures: TO / END

```
TO NAME :p1 :p2 ... :pK
  body line 1
  body line 2
  ...
END
```

Defines a procedure. **10 procs** of up to **224 bytes** of body each. Up to **2 named parameters** per proc (`:p1` `:p2`).

Calling

Just type the name:

```
NAME 10 20
```

Each argument is evaluated and bound to the corresponding parameter. Inside the body, `:p1` and `:p2` resolve to those values.

Tail recursion is free

If a proc's last statement is a recursive call, the interpreter **reuses the current frame** — no stack growth. So `SPIRAL` (below) runs ~50 logical levels in a single frame:

```
TO SPIRAL :SIZE :ANGLE
  IF :SIZE > 100 [ STOP ]
  FORWARD :SIZE
  RIGHT :ANGLE
  SPIRAL :SIZE + 2 :ANGLE
END

SPIRAL 0 90
```

Non-tail recursion

When a proc calls another proc inside `REPEAT` or before the last statement, a **control frame** is pushed (64 bytes). The control stack is 16 frames deep, so `THING1 = REPEAT 4 [THING]` and similar nested patterns work freely.

Procedure scope

Parameters live in dedicated slots that take precedence over global variables of the same name. After the call returns, your global variables are intact.

```
MAKE SIZE 20          ; global
TO BOX :SIZE
  REPEAT 4 [ FD :SIZE TR 90 ]
END
BOX 50                ; uses parameter (50)
PRINT :SIZE           ; still 20
```

Aborting / clearing

There is no `FORGET NAME` in V2.6 — defining a proc with the same name overwrites it. To reset to a clean slate, `BYE` and re-launch.

8. Dynamic-turtle sprites: SETSHAPE

```
SETSHAPE "NAME
```

(Note the leading double-quote — it's the LOGO syntax for a literal word.)

Swaps the on-screen turtle for a 16×16 hardware sprite. Available shapes:

Locomotion: - `BIRD1`, `BIRD2` — wings up / down (alternate for flapping) - `TURTL` — chunky turtle - `BOAT` — small sailboat

Narrator emotes (used by DEM2): - NORMAL , SHADES , SAD , UPSET , SICK , SUPER , GRUMPY , HAPPY , PIRATE , SLEEP , PERV , ANGRY

Sentinel: - ARROW — switches back to the bitmap-triangle turtle (the default)

The first non- ARROW SETSHAPE flips the VDP into 16×16-sprite mode and erases any visible bitmap arrow so the two render paths don't ghost each other. Going back to ARROW re-enables the bitmap path.

The sprite tint follows pen_color (set via SETPC) — see §4.

Animating a flap

```
TO FLY
  SETSHAPE "BIRD1
  FD 4
  TR 12
  PAUSE 1           ; ~100 ms hold
  SETSHAPE "BIRD2
  FD 4
  TR 12
  PAUSE 1
END

PU                 ; pen up so we don't trail the flight
HOME
REPEAT 30 [ FLY ]
```

PAUSE 1 is roughly 100 ms at 1 MHz — long enough to register both wing positions visually. Without it, each frame is ~1 ms and the wings blur.

9. Bitmap text: LABEL / SAY

The interpreter embeds the 1 024-byte Apple-1 charmap so it can render text directly into the TMS9918 bitmap (separate from the Apple-1 character display).

LABEL

```
LABEL "TEXT
```

Print TEXT (multi-word, until CR or]) on the bitmap starting at the current turtle position. 8×8 glyphs from roms/charmap.rom . Each glyph advances tx by 8.

SAY

```
SAY "TEXT
```

Comic-book speech bubble. Draws a 240×32 px frame at (8, 80) - (247, 112) with a triangular tail pointing up to (128, 72) , then prints up to **3 lines of 28 chars** wrapped automatically

(no word-boundary detection — text breaks mid-word at the right margin). Lines past the bottom are silently truncated. The pause length is proportional to the number of lines (~2.4 s per line at 1× speed).

The bubble outline and glyph cells are forced to **white** while rendering, then `pen_color` is restored. So a green sprite + a SAY shows a white bubble framed around a green character.

LIST

```
LIST [NAME]
```

Dump the body of proc `NAME` (or all procs if no argument) to the bitmap. Useful for "show me what I just typed" sessions.

EDIT

```
EDIT NAME
```

Open a fullscreen visual editor for proc `NAME`'s body. Shortcuts:

Key	Action
Ctrl-K	move cursor up
Ctrl-J	move cursor down
(printable)	overwrite at cursor
Ctrl-X	save and exit
Ctrl-Q	abort

After saving, the proc is the new body and the REPL is back. The classic ARROW turtle is restored.

10. Built-in demos

```
DEMO
```

Bitmap-turtle slideshow. Each scene picks its own `SETPC` :

`STAR`, `SUN`, `ROSETTE`, `RANDOM`, `FLOWER` (defines + calls `SQUARE` / `FLOWER`), `SPIRAL91` (defines + calls `SPIRAL`, deep tail recursion), `HEXAGON`, `STAR7`, `BURST`, `PINWHL`, `KALEID`, `GARDEN`, `CIRCLES`, `RAYS`, `BIRDFLY` (figure-8 sprite flight), then resets the arrow turtle and every colour to white before printing `END`.

The procs `SQUARE`, `FLOWER`, `SPIRAL`, `BFR`, `BFL`, `BFLY` are **defined live** inside the demo — they survive in the REPL afterwards, so you can call `FLOWER` again, `SPIRAL 0 91`, etc.

```
DEM2
```

Narrator-mode slideshow. POM1 (the dreamer in the emulator) tells its own life story through the 12 emote sprites + bitmap-text bubbles. The **ill** scene tints the sprite green; the **angry** scene tints it red. Bubble text stays white throughout.

11. Tutorials by example

Each tutorial assumes a fresh REPL (or just a `CS` to clear).

11.1 — Polygons in one line

A regular N-gon is `REPEAT N [FD step TR (360/N)]`:

```
CS
SETPC 5
REPEAT 6 [ FD 45 TR 60 ]           ; hexagon
```

Replace `6` with any divisor of 360. For $360/N$ to be an integer, common N s are 3, 4, 5, 6, 8, 9, 10, 12, 18, 24, 30, 36, 60, 72, 90.

11.2 — Stars (non-convex polygons)

A star is what you get when you turn by **more** than $360/N$:

```
CS
SETPC 11
REPEAT 7 [ FD 70 TR 154 ]          ; 7-point star,  $154 \approx 360 - 360/7 + \epsilon$ 
```

```
CS
SETPC 9
REPEAT 9 [ FD 70 TR 160 ]          ; "burst" 9-pointed
```

11.3 — A square procedure with a parameter

```
TO SQUARE :SIZE
  REPEAT 4 [ FD :SIZE TR 90 ]
END

CS
SETPC 3
SQUARE 30
TR 90
SQUARE 50
TR 90
SQUARE 70
```

Three squares of growing size, each rotated 90° from the last. Try calling `SQUARE 100` directly — the proc lives at the REPL until `BYE`.

11.4 — A recursive flower

```

TO SQUARE
  REPEAT 4 [ FD 50 TR 90 ]
END

TO FLOWER
  REPEAT 36 [ TR 10 SQUARE ]
END

CS
SETPC 13                      ; magenta
FLOWER

```

`FLOWER` calls `SQUARE` 36 times, rotating the canvas 10° between each. `SQUARE` runs at every step using a *non-tail* call (the call is followed by another iteration of `REPEAT`). This pushes one control frame for the duration of `SQUARE` and pops it when `SQUARE` returns, so the 16-frame stack handles arbitrary nesting.

11.5 — Tail-recursive spiral (no stack growth)

```

TO SPIRAL :SIZE :ANGLE
  IF :SIZE > 100 [ STOP ]
  FORWARD :SIZE
  RIGHT :ANGLE
  SPIRAL :SIZE + 2 :ANGLE
END

CS
SETPC 7                      ; cyan
SPIRAL 0 90                  ; classic right-angle spiral

```

About 50 levels of recursion in a single frame. Try `SPIRAL 0 91` for a beautiful "spiral that just barely closes" curve, or `SPIRAL 0 144` for a star-shaped spiral.

11.6 — IFELSE

```

TO COIN
  IFELSE RANDOM 2 = 0 [ PRINT "HEADS ] [ PRINT "TAILS ]
END

REPEAT 10 [ COIN ]

```

Ten coin flips, printed to the Apple-1 console.

11.7 — Dynamic-turtle bird flight (figure 8)

```

SETPC 11                                ; yellow bird

TO BFR
  SETSHAPE "BIRD1
  FD 3
  TR 12
  PAUSE 1
  SETSHAPE "BIRD2
  FD 3
  TR 12
  PAUSE 1
END

TO BFL
  SETSHAPE "BIRD1
  FD 3
  TL 12
  PAUSE 1
  SETSHAPE "BIRD2
  FD 3
  TL 12
  PAUSE 1
END

TO BFLY
  REPEAT 8 [ BFR ]
  REPEAT 8 [ BFL ]
END

PU                                      ; no trail during flight
HOME
BFLY
SETSHAPE "ARROW                        ; restore the bitmap arrow
PD

```

Each `BFR` / `BFL` is one wing-flap cycle that nets $+24^\circ$ / -24° of turn. 8 of each makes a 192° lobe — slightly more than a half-circle, giving the figure-8 its bulged shape.

11.8 — Coloured speech bubble

```

CS
SETXY 128 64
SETPC 3                                ; sprite turns green
SETSHAPE "SICK
SAY "I AM ILL.
SETPC 9                                ; sprite turns red
SETSHAPE "ANGRY
SAY "EMULATION IS NOT LIFE!
SETPC 15                               ; back to white

```

Even though `SETPC` colours the sprite, the bubble text stays white because `SAY` overrides for its rendering window.

11.9 — Random-direction rays

```
CS
SETPC 8
REPEAT 36 [ PU HOME PD SETH RANDOM 250 FD 70 ]
```

36 rays from centre, each in a random direction.

11.10 — Combining everything

```
TO PETAL :LEN
  REPEAT 18 [ FD :LEN TR 10 FD :LEN TR 170 ]
END

TO MEADOW :N
  IF :N = 0 [ STOP ]
  PU
  SETXY RANDOM 200 RANDOM 150
  PD
  SETPC RANDOM 14 + 1
  SETH RANDOM 250
  PETAL 8
  MEADOW :N - 1
END

CS
MEADOW 12
```

12 random-coloured petals scattered across the screen. Demonstrates parameters, recursion (tail), RANDOM, SETXY, SETPC, SETH, inner REPEAT.

12. Reference card

Turtle

FD n / FORWARD n	; move n pixels
BK n / BACK n	; move n pixels backward
TR n / RIGHT n	; turn right n degrees
TL n / LEFT n	; turn left n degrees
PU / PENUP	; pen up
PD / PENDOWN	; pen down
HOME	; (128, 96), heading 0
CS / CLEARSCREEN	; clear bitmap + HOME
SETXY x y	; jump to (x, y)
SETH n	; absolute heading

Colour

SETPC n	; n in 0..15, drives every surface
---------	------------------------------------

Console

```
PRINT "WORD          ; print a word
PRINT N              ; print a number
WAIT N               ; wait N "units" (~0.6 s each)
PAUSE N              ; short wait, ~0.1 s per N
BYE                  ; back to Wozmon
HELP [N]             ; help, optional page 1..8
```

Control flow

```
REPEAT N [ ... ]
REPEAT FOREVER [ ... ]      ; ESC or Ctrl-G aborts
IF a OP b [ ... ]
IFELSE a OP b [yes] [no]
STOP                        ; exit current proc
```

Operators: < > = <= >= <>

Variables and arithmetic

```
MAKE NAME N
:NAME                ; read
RANDOM N              ; 0..N-1
arg = literal | :NAME | (literal|:NAME) (op) (literal|:NAME)
op = + - * /          ; single level only
```

Procedures

```
TO NAME :p1 :p2
  body
END
NAME a b              ; call
```

Sprites (dynamic turtle)

```
SETSHAPE "NAME          ; one of:
  BIRD1 BIRD2 TURTLE BOAT
  NORMAL SHADES SAD UPSET SICK SUPER
  GRUMPY HAPPY PIRATE SLEEP PERV ANGRY
  ARROW                  ; reverts to bitmap turtle
```

Bitmap text

```
LABEL "TEXT            ; at current turtle pos
SAY "TEXT               ; speech bubble (3 lines, 28 chars wide)
LIST [NAME]             ; dump proc body to bitmap
EDIT NAME               ; visual line editor
```

Demos

DEMO	; bitmap turtle slideshow (~30 scenes)
DEM2	; narrator slideshow

13. Limitations

V2.6 ships at **99.6 % of its 16 KB ROM cap**. This dictates many of the limits below.

Language

- **2 named parameters** per procedure. Workaround: pack into vars.
- **6-character identifier limit** for vars and procs (`THING1` , `THING7` are distinct; `LONGNAME` and `LONGNAMA` collide).
- **Single arithmetic op** per argument — `:a + :b * 2` is not parsed as a tree. Decompose with intermediate `MAKE s`.
- **No negative number literals** — use `BK` for backward moves.
- **No string variables / lists** — there is no first-class word / list type beyond `PRINT "WORD` .
- **No FORGET NAME** — re-defining overwrites; cleaner state needs `BYE` then re-launch.

Storage caps

Resource	V2.6 cap
User variables	12
User procs	10
Proc body length	224 bytes
Control stack (non-tail recursion)	16 frames (1024 B)
Line buffer (single REPL line)	60 chars
<code>RANDOM N</code> argument	1..255 (it caps internally at 8-bit)
TMS9918 bitmap	256×192 px
Sprite-0 turtle size	16×16 px

Performance

- Bitmap-arrow movement: ~12 000 cycles (= ~12 ms at 1 MHz). Trail-only movement: ~3 000 cycles per pixel of trail. A 100-px `FD 100` takes about a third of a second visible (perceptible).
- `RANDOM N` is fast (LFSR step + reduction), microseconds.
- Sprite movement: ~50 cycles (4-byte attribute write).
- A `REPEAT 10000 [...]` loop is **not** instantaneous — interpreter overhead per iteration is ~200 cycles.

Hardware

- Requires the P-LAB TMS9918 Graphic Card (Mode 2).
- CodeTank daughterboard plugged into the TMS9918 card. The interpreter uses the upper Apple-1 RAM bank (\$E000-\$EFFF) for PROCBSS .
- Cannot coexist with cards that share the \$CC00 / \$CC01 window (e.g. A1-AUDIO SE) — see CLAUDE.md "one board at a time" rule for POM1's preset-level mutexes.

What V2.6 does NOT have

- **Floating-point arithmetic** — integer 16-bit only.
- **Word lists, strings, records** — no WORD , LIST , BUTFIRST , FIRST , etc.
- **Dynamic dispatch / RUN** — you cannot evaluate a string as code.
- **Files / I/O** beyond the keyboard / TMS9918 / Apple-1 console.
- **Multiple sprites** — only sprite-0 is used.
- **Sprite-position updates without redraw lag** of the bitmap arrow for the static (ARROW) case (since save/restore is ~12 000 cycles per move).

14. Internals (for the curious)

How the bitmap arrow works (option B, V2.6)

Earlier revisions drew the bitmap-triangle arrow with **XOR over the trail** — fast (idempotent restore by drawing twice), but it inverted trail bits while the arrow was visible and lost a corner pixel from each vertex (the line walker plotted shared endpoints twice, XOR'd even-count = invisible).

V2.6 ditched XOR for **save/restore** (option B):

1. Compute the 3 triangle vertices from the heading (compute_turtle_verts).
2. Snapshot a **4×4 cell (32×32 px) bounding box** anchored on (tx_lo - 9, ty_lo - 9) snapped to the cell grid, clamped to the 256×192 screen. 16 cells × 8 pattern bytes = **128 bytes** saved.
3. Draw the 3 lines with plot_set in OR mode. Trail bits are *added to* (not toggled by) the arrow.
4. On erase / move: write the 128 saved bytes back verbatim. The bitmap is restored bit-for-bit.

Why 4×4 and not 3×3: the tip extends ±9 px from (tx, ty) in any direction, and a 24 px box anchored on the cell grid can leave the tip 1 cell outside the saved region when the turtle sits in the second half of a cell — which produced the V2.5 "leftover pixel at the tip" bug. 32 px (4 cells) covers ±15 px and clears the whole range.

The save and restore use a single shared body (arrow_io_bbox) selected by an arrow_io_dir flag, with the two entry points using the classic 6502 LDA #imm / .byte \$2C / LDA #imm BIT-skip trick.

Why SETPC affects sprite-0 colour too

The TMS9918 stores sprite-0's colour in the sprite attribute table (VRAM \$3B00..\$3B03 for sprite 0: Y, X, name, colour). `draw_turtle` re-emits these 4 bytes every time the turtle moves, reading `pen_color` for the colour byte. So `SETPC` changing `pen_color` naturally propagates to the sprite at the next draw — and `cmd_setpc` calls `draw_turtle` immediately so the change is visible without waiting for the next move.

Why SAY forces white

The bitmap text glyphs are written by `blit_glyph`, which writes both the pattern bytes and the colour-table bytes. If `pen_color` were green (because the sprite is "ill"), the bubble glyphs would be green too — unreadable on a busy bitmap. `cmd_say` swaps `pen_color = 15` (white) for the duration of `draw_bubble + cmd_label`, then restores the caller's `pen_color`.

Why is the line buffer only 60 chars?

The 6502's address modes make 256-byte buffers cheap and 128-byte buffers cheaper. The interpreter's parser scratch lives in the LBUF zone (\$0200-\$027F, 128 B total), which also holds `tx0..ty2`, `plot_mode`, `turtle_visible`, etc. The line buffer is what's left: 60 chars + null terminator + counters. The 4 extra bytes that V2.6's arrow save/restore needs (`arrow_rows_left`, `arrow_cols_left`, `arrow_bx`, `arrow_io_dir`) ate into that margin — which is why the 128-byte arrow pattern buffer itself lives in PROCBSS, not LBUF.

15. Hardware and Memory map

V2.6 layout (CodeTank, jumper Upper):

```
$0000-$001F  ZP scalars (math + tms9918m2 imports)
$0020-$00FF  ZP free
$0100-$01FF  6502 stack
$0200-$027F  LBUF: line buffer + parser scratch
$0280-$0FFF  free Apple-1 RAM (programmer's playground)
$4000-$7FFF  CODE: 16 KB ROM, interpreter linked + run-in-place
$E000-$EFFF  PROCBSS: control stack + var/proc tables + arrow save buf
               (upper Parmigiani RAM bank, 4 KB)
```

TMS9918 VRAM layout (Mode 2):

```
$0000-$17FF  pattern table (6 144 B = 256x192 / 8)
$1800-$1FFF  sprite pattern table
$2000-$37FF  colour table (6 144 B, 8x8 cells)
$3800-$3AFF  name table
$3B00-$3B7F  sprite attribute table (32 sprites x 4 bytes)
```

I/O (TMS9918):

```
$CC00  VDP_DATA (read/write VRAM at the cursor)
$CC01  VDP_CTRL (set VRAM cursor / register writes)
```

Apple-1:

```
$D010  KBD      last key, bit 7 = 1 means new key
$D011  KBDCR    bit 7 = 1 when key ready
$D012  DSP      write = display char
$E000  Integer BASIC ROM (untouched by LOGO)
$FF00  Woz Monitor (BYE returns here)
```

End of manual. Happy turtling.

— VERHILLE Arnaud, 2026