

# APPLE-1 LOGO V2.6 — Manuel

---

*Interpréteur LOGO pour la carte graphique P-LAB TMS9918 sur Apple-1*

(C) 2026 VERHILLE Arnaud

---

## Table des matières

---

1. Qu'est-ce qu'APPLE-1 LOGO ?
  2. Installation et premier REPL
  3. La tortue
  4. Couleur
  5. Répétition et contrôle de flux
  6. Variables et arithmétique
  7. Procédures
  8. Sprites
  9. Texte sur le bitmap
  10. Démonstrations intégrées
  11. Tutoriels par l'exemple
  12. Fiche de référence
  13. Limites
  14. Internes (pour les curieux)
  15. Matériel et plan mémoire
- 

## 1. Qu'est-ce qu'APPLE-1 LOGO ?

---

APPLE-1 LOGO V2.6 est un interpréteur LOGO complet qui tourne sur un vrai Apple-1 (ou l'émulateur POM1) équipé de la **carte graphique P-LAB TMS9918**. Il dessine sur le bitmap 256×192 (Mode 2) du TMS9918 en utilisant les sprites matériels de la puce pour les tortues animées.

C'est un *vrai* LOGO avec :

- Une tortue qui répond à `FORWARD`, `RIGHT`, `LEFT`, `HOME`, etc.
- Des procédures utilisateur avec paramètres et récursion.
- Des variables et un niveau d'arithmétique en position d'argument.
- `IF` / `IFELSE` / `STOP`, `REPEAT`, `REPEAT FOREVER` (interrompable par ESC).
- Une commande couleur unifiée, des sprites tortue, du texte bitmap.
- Un éditeur plein-écran pour les procédures (`EDIT`) et un dumper de procs (`LIST`).
- Deux diaporamas (`DEMO`, `DEM2`) pour montrer les capacités.

Source : `TMS_Logo_16k.asm`, lié à `$4000` et livré sur la fille CodeTank (banc supérieur). Basculer le jumper CodeTank sur **Upper** et lancer l'interpréteur avec `4000R` depuis Wozmon.

---

## 2. Installation et premier REPL

---

### Dans POM1

1. Lancer POM1 et choisir le **Preset 2** (P-LAB Apple-1 with TMS9918 + CodeTank daughterboard).
2. Basculer le jumper CodeTank sur **Upper** dans le menu Hardware, puis dans Wozmon (le prompt `\`) taper : `4000R`
3. L'écran TMS9918 s'ouvre, la tortue (un petit triangle) apparaît au centre, et le prompt attend : `APPLE-1 LOGO FOR TMS9918 - TYPE HELP (C) 2026 VERHILLE ARNAUD ?`

### Sur un vrai Apple-1

Il faut un Apple-1, la carte P-LAB TMS9918 et la fille CodeTank. Flasher `Codetank_GAME1.rom` dans la 28C256, mettre le jumper sur Upper, puis taper `4000R` depuis Wozmon.

### Dire bonjour

Essayez :

```
PRINT "HELLO
```

La console répond `HELLO` . Le prompt `?` revient.

Essayez un mouvement de tortue :

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

Deux tracés perpendiculaires de 60 pixels chacun.

Toujours disponible :

```
HELP
```

imprime la table des matières. `HELP 1` à `HELP 8` paginent les sections (tortue, console, variables, procs, tortue dynamique, démos, erreurs).

`BYE` rend la main à Wozmon.

---

## 3. La tortue

---

La tortue a une position `(x, y)` en coordonnées pixel (0..255 × 0..191, origine en haut-à-gauche), un cap en degrés (0 = nord, 90 = est), et un état du stylo (baissé / levé).

## Mouvement

Commande	Alias	Effet
FD n	FORWARD n	avance de n pixels dans le cap courant ; trace si stylo baissé
BK n	BACK n	recule de n pixels
TR n	RIGHT n	tourne à droite (sens horaire) de n degrés
TL n	LEFT n	tourne à gauche de n degrés
SETXY x y		saute à (x, y)
SETH n		fixe le cap à n degrés (mod 360)
HOME		va au centre (128, 96) , cap 0, état du stylo préservé

n est un entier non signé 0..65535. Les négatifs ne sont pas parsés ; utiliser BK au lieu de FD -n .

## Stylo

Commande	Alias	Effet
PD	PENDOWN	la tortue trace
PU	PENUP	la tortue se déplace en silence

HOME préserve l'état du stylo — un PU + HOME garde le stylo levé pour que les mouvements suivants ne tracent pas une ligne vers l'origine.

## Écran

Commande	Alias	Effet
CS	CLEARSCREEN	efface le bitmap (pixels seulement, pas la table de couleurs) ; HOME implicite

## Une première image

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

Un carré de 50 pixels, dessiné depuis le centre.

## 4. Couleur : la commande unique SETPC

V2.6 a exactement une commande couleur :

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

Elle pilote **toutes** les surfaces colorisées simultanément :

- la **traînée** de la tortue ;
- la **flèche bitmap** (la forme ARROW par défaut) — ses cellules de contour sont repeintes en direct ;
- la **tortue sprite-0** (toute autre valeur de SETSHAPE) — l'attribut sprite est ré-émis avec la nouvelle couleur ;
- le **texte bitmap** dessiné par LABEL et l'éditeur visuel ( LIST / EDIT ).

La palette TMS9918 (Mode 2) :

n	Couleur	n	Couleur
0	transparent	8	rouge moyen
1	noir	9	rouge clair
2	vert moyen	10	jaune foncé
3	vert clair	11	jaune clair
4	bleu foncé	12	vert foncé
5	bleu clair	13	magenta
6	rouge foncé	14	gris
7	cyan	15	blanc ( <i>défaut</i> )

SAY (la bulle de BD, voir §9) est la seule exception — il force le contour de la bulle et les cellules glyphes en blanc pour la lisibilité, puis restaure votre pen\_color pour que les commandes suivantes gardent la teinte choisie.

### Exemples

```
SETPC 11          ; jaune
REPEAT 5 [FD 80 RIGHT 144]      ; étoile à 5 branches jaune
```

```
SETPC 9           ; rouge
SETSHAPE "ANGRY ; le sprite devient rouge à la volée
```

Les anciennes commandes V2.5 SETTC / SETSC ont disparu.

## 5. Répétition et contrôle de flux

---

### REPEAT

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

Exécute le bloc entre crochets **N** fois. Les blocs peuvent contenir n'importe quelle commande, y compris d'autres **REPEAT** et des appels de proc.

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

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

### REPEAT FOREVER

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

Boucle jusqu'à appui sur **ESC** ou **Ctrl-G**. L'interpréteur abandonne proprement : les procédures définies survivent.

### IF / IFELSE

```
IF a OP b [ ... ]  
IFELSE a OP b [ bloc-oui ] [ bloc-non ]
```

**OP** est l'un de **<**, **>**, **=**, **<=**, **>=**, **<>** (différent). **a** et **b** peuvent être des nombres littéraux ou des références **:VARNAME**.

```
IF :SIZE > 100 [ STOP ] ; terminaison classique de spirale  
IFELSE :N = 0 [ PRINT "ZERO ] [ PRINT "NONZERO ]
```

### STOP

**STOP** quitte la procédure courante immédiatement. Dans un **REPEAT**, la boucle se déroule et **STOP** se propage à la procédure englobante.

### Sortir

```
BYE
```

Retour à Wozmon (si LOGO a été lancé depuis là).

---

## 6. Variables et arithmétique

---

### MAKE

```
MAKE NAME N
```

Crée ou met à jour une variable `NAME` avec la valeur `N`. `NAME` fait au plus **6 caractères** (A-Z, 0-9). Pas de soulignés en tête.

### Lecture

`:NAME` s'évalue à la valeur de la variable.

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

### Arithmétique en position d'argument

Chaque argument supporte **un seul** opérateur :

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

Opérateurs autorisés : `+` `-` `*` `/`. Multiplication et division en 16 bits non signés. Pas d'imbrication : `:a + :b + 1` n'est **pas** parsé comme une chaîne — utilisez un `MAKE` temporaire.

### RANDOM

```
RANDOM N
```

Renvoie un nombre dans `0..N-1`. Implémenté par un LFSR 16-bit amorcé au boot.

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

### Limites

- 12 variables nommées max (la table est petite pour laisser de la place aux procs).
- Limite de 6 caractères sur les noms — `THING` et `THING1` restent distincts, mais `LONGNAME` et `LONGNAMA` collisionnent sur les 6 premiers caractères.

## 7. Procédures : TO / END

```
TO NAME :p1 :p2 ... :pK
  ligne corps 1
  ligne corps 2
  ...
END
```

Définit une procédure. **10 procs** de **224 octets** de corps maximum chacune. Jusqu'à **2 paramètres** nommés par proc ( :p1 :p2 ).

### Appel

Tapez simplement le nom :

```
NAME 10 20
```

Chaque argument est évalué et lié au paramètre correspondant. Dans le corps, :p1 et :p2 se résolvent en ces valeurs.

### La récursion terminale est gratuite

Si la dernière instruction d'une proc est un appel récursif, l'interpréteur **réutilise le frame courant** — pas de croissance de pile. Donc SPIRAL (ci-dessous) tourne ~50 niveaux logiques en une seule frame :

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

SPIRAL 0 90
```

### Récursion non-terminale

Quand une proc appelle une autre proc dans un REPEAT ou avant la dernière instruction, un **frame de contrôle** est empilé (64 octets). La pile de contrôle fait 16 frames de profondeur, donc des motifs imbriqués comme THING1 = REPEAT 4 [ THING ] fonctionnent librement.

### Portée

Les paramètres vivent dans des slots dédiés qui priment sur les variables globales du même nom. Au retour de l'appel, vos variables globales sont intactes.

```
MAKE SIZE 20          ; globale
TO BOX :SIZE
  REPEAT 4 [ FD :SIZE TR 90 ]
END
BOX 50                ; utilise le paramètre (50)
PRINT :SIZE           ; toujours 20
```

## Annulation / nettoyage

Pas de `FORGET NAME` en V2.6 — redéfinir avec le même nom écrase. Pour repartir d'un état propre, `BYE` puis relancer.

## 8. Tortue dynamique : SETSHAPE

```
SETSHAPE "NAME
```

(Notez le double-quote en tête — c'est la syntaxe LOGO pour un mot littéral.)

Échange la tortue à l'écran pour un sprite matériel 16×16. Formes disponibles :

**Locomotion** : - `BIRD1` , `BIRD2` — ailes hautes / basses (alternance pour le battement) - `TURTL` — tortue trapue - `BOAT` — petit voilier

**Émotes du narrateur** (utilisés par `DEM2`) : - `NORMAL` , `SHADES` , `SAD` , `UPSET` , `SICK` , `SUPER` , `GRUMPY` , `HAPPY` , `PIRATE` , `SLEEP` , `PERV` , `ANGRY`

**Sentinelle** : - `ARROW` — bascule en mode flèche bitmap (le défaut)

Le premier `SETSHAPE` non-`ARROW` bascule le VDP en mode sprite 16×16 et efface toute flèche bitmap visible pour que les deux chemins de rendu ne se télescopent pas. Repasser à `ARROW` réactive le chemin bitmap.

La teinte du sprite suit `pen_color` (réglée via `SETPC`) — voir §4.

## Animer un battement d'ailes

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

PU                ; stylo levé pour ne pas tracer le vol
HOME
REPEAT 30 [ FLY ]
```

PAUSE 1 fait environ 100 ms à 1 MHz — assez long pour voir les deux positions d'ailes. Sans, chaque frame fait ~1 ms et les ailes se brouillent.

---

## 9. Texte sur le bitmap : LABEL / SAY

---

L'interpréteur embarque la charmap Apple-1 de 1 024 octets pour rendre du texte directement dans le bitmap TMS9918 (séparé de l'affichage caractère Apple-1).

### LABEL

```
LABEL "TEXT"
```

Imprime TEXT (multi-mots, jusqu'au CR ou ] ) sur le bitmap à la position courante de la tortue. Glyphes 8×8 issus de roms/charmap.rom. Chaque glyphe avance tx de 8 pixels.

### SAY

```
SAY "TEXT"
```

Bulle de bande dessinée. Dessine un cadre 240×32 px à (8, 80)-(247, 112) avec une queue triangulaire pointant vers (128, 72), puis imprime jusqu'à **3 lignes de 28 caractères** wrappées automatiquement (pas de détection de frontière de mot — le texte casse au milieu des mots à la marge droite). Les lignes au-delà du bas sont silencieusement tronquées. La pause est proportionnelle au nombre de lignes (~2,4 s par ligne en vitesse 1×).

Le contour de la bulle et les cellules glyphes sont forcés en **blanc** pendant le rendu, puis pen\_color est restauré. Donc un sprite vert + un SAY donne une bulle blanche autour d'un personnage vert.

### LIST

```
LIST [NAME]
```

Dump le corps de la proc NAME (ou toutes les procs si pas d'argument) sur le bitmap. Utile pour les sessions « montre-moi ce que je viens de taper ».

### EDIT

```
EDIT NAME
```

Ouvre un éditeur visuel plein écran pour le corps de NAME. Raccourcis :

Touche	Action
Ctrl-K	curseur vers le haut
Ctrl-J	curseur vers le bas
(caractère imprimable)	écrase à la position du curseur
Ctrl-X	enregistrer et sortir
Ctrl-Q	abandonner

Après enregistrement, la proc a son nouveau corps et le REPL revient. La flèche ARROW classique est restaurée.

## 10. Démonstrations intégrées

### DEMO

Diaporama tortue bitmap. Chaque scène choisit son SETPC :

STAR, SUN, ROSETTE, RANDOM, FLOWER (définit + appelle SQUARE / FLOWER), SPIRAL91 (définit + appelle SPIRAL, récursion terminale profonde), HEXAGON, STAR7, BURST, PINWHL, KALEID, GARDEN, CIRCLES, RAYS, BIRDFLY (vol en figure-8), puis réinitialise la flèche tortue et toutes les couleurs en blanc avant d'imprimer END.

Les procs SQUARE, FLOWER, SPIRAL, BFR, BFL, BFLY sont **définies en direct** dans la démo — elles survivent dans le REPL ensuite, vous pouvez relancer FLOWER, SPIRAL 0 91, etc.

### DEM2

Diaporama mode narrateur. POM1 (le rêveur dans l'émulateur) raconte sa propre histoire à travers les 12 sprites émotes + bulles de texte bitmap. La scène **iii** teinte le sprite en vert ; la scène **angry** en rouge. Le texte des bulles reste blanc tout du long.

## 11. Tutoriels par l'exemple

Chaque tutoriel suppose un REPL frais (ou juste un CS pour nettoyer).

### 11.1 — Polygones en une ligne

Un N-gone régulier est REPEAT N [FD pas TR (360/N)] :

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

Remplacez 6 par un diviseur de 360. Pour que 360/N soit entier, les N habituels sont 3, 4, 5, 6, 8, 9, 10, 12, 18, 24, 30, 36, 60, 72, 90.

## 11.2 — Étoiles (polygones non-convexes)

Une étoile c'est ce qu'on obtient en tournant de **plus** que  $360/N$  :

```
CS
SETPC 11
REPEAT 7 [ FD 70 TR 154 ]           ; étoile à 7 branches,  $154 \approx 360 - 360/7 + \varepsilon$ 
```

```
CS
SETPC 9
REPEAT 9 [ FD 70 TR 160 ]          ; « burst » à 9 pointes
```

## 11.3 — Procédure carré avec un paramètre

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

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

Trois carrés de tailles croissantes, chacun tourné de  $90^\circ$  par rapport au précédent. Essayez d'appeler `SQUARE 100` directement — la proc vit dans le REPL jusqu'à `BYE`.

## 11.4 — Une fleur récursive

```
TO SQUARE
  REPEAT 4 [ FD 50 TR 90 ]
END

TO FLOWER
  REPEAT 36 [ TR 10 SQUARE ]
END

CS
SETPC 13           ; magenta
FLOWER
```

`FLOWER` appelle `SQUARE` 36 fois en tournant le canevas de  $10^\circ$  entre chaque. `SQUARE` tourne à chaque pas via un appel *non-terminal* (l'appel est suivi d'une autre itération de `REPEAT`). Cela empile un frame de contrôle pour la durée de `SQUARE` et le dépile au retour, donc la pile à 16 frames gère n'importe quel imbrication.

## 11.5 — Spirale tail-réursive (sans croissance de pile)

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

CS
SETPC 7 ; cyan
SPIRAL 0 90 ; spirale classique à angle droit
```

Environ 50 niveaux de récursion en une seule frame. Essayez `SPIRAL 0 91` pour une superbe « spirale qui se referme à peine », ou `SPIRAL 0 144` pour une spirale en forme d'étoile.

## 11.6 — IFELSE

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

REPEAT 10 [ COIN ]
```

Dix lancers de pièce, imprimés sur la console Apple-1.

## 11.7 — Vol d'oiseau dynamique (figure 8)

```

SETPC 11                                ; oiseau jaune

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                                      ; pas de traînée pendant le vol
HOME
BFLY
SETSHAPE "ARROW                        ; restaurer la flèche bitmap
PD

```

Chaque `BFR` / `BFL` est un cycle de battement qui tourne net de  $+24^\circ$  /  $-24^\circ$ . 8 de chaque font une boucle de  $192^\circ$  — un peu plus qu'un demi-cercle, ce qui donne au figure-8 sa forme ventrue.

## 11.8 — Bulle colorée

```

CS
SETXY 128 64
SETPC 3                                ; le sprite devient vert
SETSHAPE "SICK
SAY "I AM ILL.
SETPC 9                                ; le sprite devient rouge
SETSHAPE "ANGRY
SAY "EMULATION IS NOT LIFE!
SETPC 15                               ; retour blanc

```

Même si `SETPC` colore le sprite, le texte de la bulle reste blanc parce que `SAY` force la couleur pour sa fenêtre de rendu.

## 11.9 — Rayons aléatoires

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

36 rayons depuis le centre, chacun dans une direction aléatoire.

## 11.10 — Tout combiner

```
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 pétales de couleurs aléatoires éparpillés sur l'écran. Démontre les paramètres, la récursion (terminale), `RANDOM`, `SETXY`, `SETPC`, `SETH`, `REPEAT` interne.

## 12. Fiche de référence

### Tortue

FD n / FORWARD n	; avance de n pixels
BK n / BACK n	; recule de n pixels
TR n / RIGHT n	; tourne à droite de n degrés
TL n / LEFT n	; tourne à gauche de n degrés
PU / PENUP	; stylo levé
PD / PENDOWN	; stylo baissé
HOME	; (128, 96), cap 0
CS / CLEARSCREEN	; efface bitmap + HOME
SETXY x y	; saute à (x, y)
SETH n	; cap absolu

### Couleur

SETPC n	; n dans 0..15, pilote toutes les surfaces
---------	--

## Console

```
PRINT "WORD           ; imprime un mot
PRINT N               ; imprime un nombre
WAIT N               ; attend N « unités » (~0,6 s chacune)
PAUSE N              ; courte attente, ~0,1 s par N
BYE                  ; retour à Wozmon
HELP [N]             ; aide, page optionnelle 1..8
```

## Contrôle de flux

```
REPEAT N [ ... ]
REPEAT FOREVER [ ... ] ; ESC ou Ctrl-G interrompt
IF a OP b [ ... ]
IFELSE a OP b [oui] [non]
STOP ; quitte la proc courante
```

Opérateurs : < > = <= >= <>

## Variables et arithmétique

```
MAKE NAME N
:NAME ; lecture
RANDOM N ; 0..N-1
arg = literal | :NAME | (literal|:NAME) (op) (literal|:NAME)
op = + - * / ; un seul niveau
```

## Procédures

```
TO NAME :p1 :p2
  corps
END
NAME a b ; appel
```

## Sprites (tortue dynamique)

```
SETSHAPE "NAME ; un parmi :
  BIRD1 BIRD2 TURTLE BOAT
  NORMAL SHADES SAD UPSET SICK SUPER
  GRUMPY HAPPY PIRATE SLEEP PERV ANGRY
  ARROW ; revient à la flèche bitmap
```

## Texte bitmap

```
LABEL "TEXT ; à la position courante de la tortue
SAY "TEXT ; bulle (3 lignes, 28 chars de large)
LIST [NAME] ; dump le corps de la proc sur le bitmap
EDIT NAME ; éditeur de ligne visuel
```

## Demos

DEMO	; diaporama tortue bitmap (~30 scènes)
DEM2	; diaporama narrateur

## 13. Limites

V2.6 est livré à **99,6 % de son cap ROM 16 Ko** dans le build CodeTank. C'est ce qui dicte la plupart des limites ci-dessous.

### Langage

- **2 paramètres nommés** par procédure. Contournement : packer dans des variables globales.
- **Limite de 6 caractères** sur les noms de variables et de procs ( `THING1` , `THING7` distincts ; `LONGNAME` et `LONGNAMA` collisionnent).
- **Une seule opération arithmétique** par argument — `:a + :b * 2` n'est pas parsé en arbre. Décomposez avec des `MAKE` intermédiaires.
- **Pas de littéraux négatifs** — utiliser `BK` pour les mouvements arrière.
- **Pas de variables string / listes** — pas de type word / list au delà de `PRINT "WORD` .
- **Pas de `FORGET NAME`** — redéfinir écrase ; pour un état propre, `BYE` puis relancer.

### Caps de stockage

Ressource	Cap V2.6
Variables utilisateur	12
Procs utilisateur	10
Longueur du corps de proc	224 octets
Pile de contrôle (récursion non-terminale)	16 frames (1024 B)
Buffer de ligne (une ligne REPL)	60 chars
Argument de <code>RANDOM N</code>	1..255 (cape interne 8-bit)
Bitmap TMS9918	256×192 px
Taille du sprite-0 tortue	16×16 px

### Performance

- Mouvement de la flèche bitmap : ~12 000 cycles (= ~12 ms à 1 MHz). Mouvement avec traînée : ~3 000 cycles par pixel de traînée. Un `FD 100` prend environ un tiers de seconde visible (perceptible).
- `RANDOM N` est rapide (LFSR + réduction), microsecondes.
- Mouvement de sprite : ~50 cycles (écriture d'attribut 4 octets).
- Une boucle `REPEAT 10000 [ ... ]` n'est **pas** instantanée — l'overhead par itération de l'interpréteur est de ~200 cycles.

## Matériel

- Requiert la carte graphique P-LAB TMS9918 (Mode 2).
- Carte fille CodeTank branchée sur la carte TMS9918. L'interpréteur utilise le banc RAM haut de l'Apple-1 ( \$E000-\$EFFF ) pour PROCBSS .
- Ne peut coexister avec des cartes qui partagent la fenêtre \$CC00 / \$CC01 (ex. A1-AUDIO SE) — voir la règle CLAUDE.md « one board at a time » pour les mutex au niveau preset de POM1.

## Ce que V2.6 N'A PAS

- **Arithmétique flottante** — entiers 16 bits seulement.
- **Listes de mots, strings, records** — pas de WORD , LIST , BUTFIRST , FIRST , etc.
- **Dispatch dynamique / RUN** — impossible d'évaluer une string comme du code.
- **Fichiers / I/O** au-delà du clavier / TMS9918 / console Apple-1.
- **Plusieurs sprites** — seul le sprite-0 est utilisé.
- **Mises à jour de position de sprite sans lag de redessin** de la flèche bitmap pour le cas statique ( ARROW ) (puisque save/restore fait ~12 000 cycles par mouvement).

## 14. Internes (pour les curieux)

### Comment fonctionne la flèche bitmap (option B, V2.6)

Les versions antérieures dessinaient la flèche-triangle bitmap par **XOR sur la traînée** — rapide (restauration idempotente en redessinant deux fois), mais ça inversait les bits de la traînée pendant que la flèche était visible et perdait un pixel de coin à chaque sommet (le tracé de Bresenham plottait les endpoints partagés deux fois, XOR à compte pair = invisible).

V2.6 a abandonné XOR pour **save/restore** (option B) :

1. Calcul des 3 sommets du triangle d'après le cap ( compute\_turtle\_verts ).
2. Snapshot d'une **boîte englobante de 4×4 cells (32×32 px)** ancrée sur (tx\_lo - 9, ty\_lo - 9) snappée au grid des cellules, clampée aux 256×192 de l'écran. 16 cells × 8 octets pattern = **128 octets** sauvés.
3. Tracé des 3 lignes avec plot\_set en mode OR. Les bits de la traînée sont *ajoutés à* (et non basculés par) la flèche.
4. À l'erase / move : on récrit les 128 octets sauvés tels quels. Le bitmap est restauré bit pour bit.

Pourquoi 4×4 et pas 3×3 : la pointe peut s'étendre à ±9 px de (tx, ty) dans n'importe quelle direction, et une boîte de 24 px ancrée sur le grid peut laisser la pointe sortir d'une cellule quand la tortue est dans la deuxième moitié d'une cellule — c'est ce qui produisait le bug V2.5 « pixel résiduel à la pointe ». 32 px (4 cells) couvre ±15 px et nettoie toute la plage.

Save et restore partagent un même corps (arrow\_io\_bbox) sélectionné par un flag arrow\_io\_dir, avec les deux entry points utilisant le classique trick 6502 LDA #imm / .byte \$2C / LDA #imm (skip BIT).

## Pourquoi SETPC affecte aussi la couleur du sprite-0

Le TMS9918 stocke la couleur du sprite-0 dans la table d'attributs sprite (VRAM \$3B00..\$3B03 pour le sprite 0 : Y, X, name, couleur). `draw_turtle` ré-émet ces 4 octets à chaque mouvement de la tortue, en lisant `pen_color` pour l'octet couleur. Donc `SETPC` qui change `pen_color` se propage naturellement au sprite au prochain dessin — et `cmd_setpc` appelle `draw_turtle` immédiatement pour que le changement soit visible sans attendre un mouvement.

## Pourquoi SAY force le blanc

Les glyphes texte bitmap sont écrits par `blit_glyph`, qui écrit à la fois les octets pattern et les octets de la table de couleurs. Si `pen_color` était vert (parce que le sprite est « malade »), les glyphes de la bulle seraient verts aussi — illisible sur un bitmap chargé. `cmd_say` swap `pen_color = 15` (blanc) pour la durée de `draw_bubble + cmd_label`, puis restaure le `pen_color` de l'appelant.

## Pourquoi le buffer de ligne ne fait que 60 chars ?

Les modes d'adressage du 6502 rendent les buffers de 256 octets bon marché et ceux de 128 octets encore moins chers. Le scratch parser de l'interpréteur vit dans la zone LBUF (\$0200-\$027F, 128 B au total), qui contient aussi `tx0..ty2`, `plot_mode`, `turtle_visible`, etc. Le buffer de ligne est ce qui reste : 60 chars + null terminator + compteurs. Les 4 octets supplémentaires que le save/restore de V2.6 demande (`arrow_rows_left`, `arrow_cols_left`, `arrow_bx`, `arrow_io_dir`) ont entamé cette marge — c'est pour ça que le buffer pattern de 128 octets de la flèche vit en PROCBSS, pas en LBUF.

# 15. Matériel et plan mémoire

Layout V2.6 (CodeTank, jumper Upper) :

```
$0000-$001F  scalaires ZP (imports math + tms9918m2)
$0020-$00FF  ZP libre
$0100-$01FF  pile 6502
$0200-$027F  LBUF : buffer de ligne + scratch parser
$0280-$0FFF  RAM Apple-1 libre (terrain de jeu du programmeur)
$4000-$7FFF  CODE : 16 Ko ROM, interpréteur lié et run-in-place
$E000-$EFFF  PROCBSS : pile de contrôle + tables var/proc + buffer save flèche
                (banc RAM Parmigiani haut, 4 Ko)
```

Plan VRAM TMS9918 (Mode 2) :

```
$0000-$17FF  table pattern (6 144 B = 256x192 / 8)
$1800-$1FFF  table pattern sprite
$2000-$37FF  table couleurs (6 144 B, cells 8x8)
$3800-$3AFF  table de noms
$3B00-$3B7F  table d'attributs sprite (32 sprites x 4 octets)
```

I/O (TMS9918) :

```
$CC00 VDP_DATA (lit/écrit VRAM au curseur)
$CC01 VDP_CTRL (set curseur VRAM / écritures de registres)
```

Apple-1 :

```
$D010 KBD      dernière touche, bit 7 = 1 quand nouvelle touche
$D011 KBDCR    bit 7 = 1 quand touche prête
$D012 DSP      écriture = char à afficher
$E000 Integer BASIC ROM (intacte par LOGO)
$FF00 Woz Monitor (BYE rend la main ici)
```

---

*Fin du manuel. Bonne tortue.*

— VERHILLE Arnaud, 2026