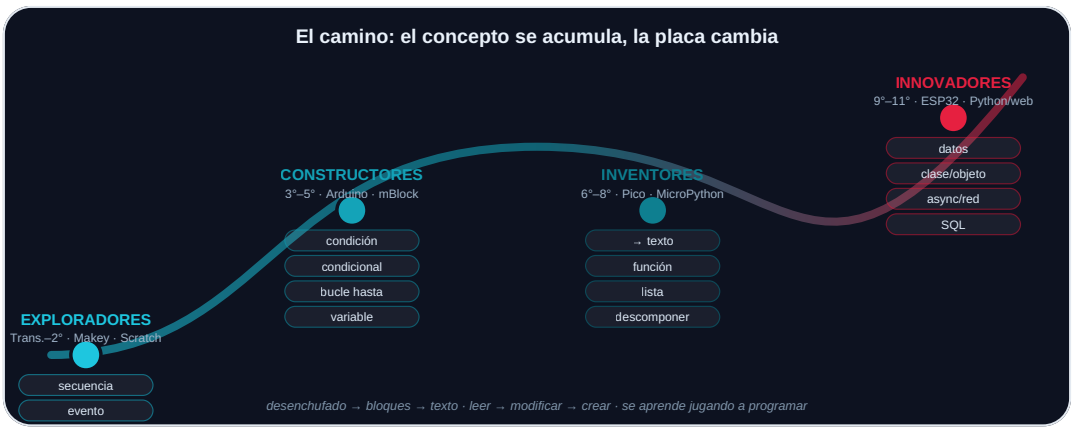
Línea Ares — Ruta de Aprendizaje de la Programación

ROBOTSchool · Documento maestro (fuente única) Versión 0.1 — Junio 2026

> La **columna vertebral**: *qué* se aprende, *en qué orden*, *con qué juego* y *cómo sabemos que se aprendió*, de Transición a 11°. Antes de escribir lecciones (doc 20) hay que fijar este camino. Está construido sobre evidencia (trayectorias de aprendizaje K-8 de Rich et al., estándares **CSTA K-12**, enfoques **PRIMM** y **Usar-Modificar-Crear**) y sobre un principio que la práctica confirma: **los niños aprenden a programar jugando — pero jugando a programar.**



1. El problema real (por qué este documento existe)

Construir pensamiento lógico y computacional en un niño es difícil: lo abstracto (una variable, un bucle, una condición) no se ve ni se toca, y la

página en blanco asusta. Un camino mal diseñado produce dos fracasos típicos:

- **Copiar sin entender:** el niño arma el robot y "funciona", pero no sabría reescribir el código. El concepto quedó tapado por el proyecto.
- **Saltos de abstracción:** se introduce un concepto antes de que exista el que lo sostiene (p. ej. "repetir hasta que…" antes de tener la idea de *condición*), y el niño se pierde.

La ruta evita ambos con cuatro decisiones de diseño.

---

2. Las cuatro decisiones de diseño del camino

1. **El camino es de conceptos, no de herramientas.** Lo que progresa es la *idea* (secuencia → bucle → condición → variable → función → datos → objetos). Las placas (Makey, Arduino, Pico, ESP32) y lenguajes solo le dan más potencia a la misma lógica. *La placa cambia; el pensamiento se acumula.*
2. **Es en espiral.** Cada concepto se revisita más profundo cada banda. Nadie "termina" los bucles en 4°: los vuelve a ver con más exigencia en 7° y en 10°.
3. **Sube por dos escaleras a la vez:**
   - *Escalera de abstracción:* **desenchufado** → **bloques** → **texto**. El mismo concepto, cada vez menos concreto.
   - *Escalera de autonomía:* **leer** → **modificar** → **crear** (PRIMM / Usar-Modificar-Crear). Nunca se empieza creando desde cero.
4. **Se aprende jugando a programar** (pilar de §4). El juego es el vehículo del concepto, y el proyecto del robot es el "jefe final" donde se aplica.

---

3. Qué se aprende y en qué orden (el grafo de dependencias)

El orden **no es arbitrario**: cada concepto necesita que otro ya esté firme. Esta es la regla que ningún libro ni lección puede romper.

Por qué este orden (rationale honesto):

- **Secuencia** antes que nada: si el orden no importa, nada importa.
- **Bucle por conteo** = abreviar una secuencia repetida (“salta 4 veces”). Es repetición *sin* condición, por eso va primero.
- **Condición** (una pregunta de verdadero/falso) debe existir **antes** del condicional y del bucle condicional, porque ambos la usan.
- **Bucle condicional** (“repite hasta que…”) combina repetición + condición → va **después** de las dos.
- **Variable** se introduce cuando *se necesita*: llevar un puntaje, contar intentos, recordar un estado. Introducirla antes es memorización vacía.
- **Función** se introduce cuando el niño ya **sintió el dolor** de repetir el mismo bloque de código. La abstracción es difícil; se gana, no se regala. Coincide con el paso de bloques a texto (Inventores).

- **Listas, datos, objetos, concurrencia** son organización de complejidad creciente — solo tienen sentido cuando hay programas grandes que organizar (Innovadores).

---

4. El juego como vehículo: *programar jugando* (pilar central)

Por qué funciona. Un buen juego de programación da lo que un niño necesita para construir lógica: una **meta clara** (ganar el nivel), **retroalimentación inmediata** (el personaje choca o avanza), **dificultad graduada** (niveles), y **repetición motivada** (quiere volver a intentar). Convierte la depuración —normalmente frustrante— en parte del juego: fallar es jugar.

Tres tipos de juego, los tres se usan:

Tipo	Qué es	Ejemplos	Para qué concepto
Programar para ganar	El niño escribe el programa que mueve un personaje/robot por un reto	Lightbot, Blockly Games, code.org, Reeborg, CodeCombat	Secuencia, bucle, condicional, función
Hacer tu propio juego	El niño *programa un juego* (no solo juega)	Scratch, MakeCode Arcade	Eventos, variables (puntaje/vidas), condicionales
El mundo/robot como juego	El reto físico o simulado es el "jefe"	Wokwi, Reeborg, el robot del proyecto	Aplicar todo lo anterior

El filtro (no todo “juego de código” sirve)

Aquí está mi advertencia como mentor: **muchos “coding games” no enseñan a programar** — son rompecabezas con código escondido o *edutainment* que entretiene sin transferir. Un juego entra a Ares solo si pasa este filtro:

1. **El niño programa de verdad** (escribe/arma instrucciones), no solo arrastra al azar hasta que pasa.
2. **El concepto es la mecánica de ganar** (no se puede ganar sin usar el bucle/condición que toca).
3. **Transfiere**: lo aprendido se puede *representar* (diagrama/pseudocódigo) y llevar a otro entorno.
4. **Tiene puente al código real.** Un juego que nunca conecta con Scratch/MicroPython/Python deja al niño atrapado en el juguete.

> **El puente es obligatorio.** Cada juego termina con: *jugaste → ahora dibújalo (diagrama) → ahora hazlo en Scratch/Python → ahora aplícalo al robot.* Sin ese puente, el juego es recreo, no aprendizaje. (Conecta con el ciclo de §7.)

Advertencia de disponibilidad y costo (para colegios)

Como esto se va a usar de verdad, verifiqué la gratuidad (junio 2026):

- **100% gratis, sin cuenta o con cuenta gratuita:** Blockly Games (Google), Scratch / ScratchJr (MIT), Code.org CS Fundamentals, Reeborg’s World, CodinGame, CheckiO, Code Club World (Raspberry Pi Foundation), Lightbot *Hour of Code*.
- **Cuidado — freemium: CodeCombat** libera solo ~11 niveles del primer capítulo gratis; lo demás es suscripción ($10–40/mes). Úsalo como *degustación*, no como columna. Tynker y CodeMonkey también son de pago en su mayoría.
- **Regla Ares:** la ruta se sostiene **solo con los 100% gratuitos**; lo freemium es opcional/complementario y nunca un requisito para avanzar.

4.1 Historias programables (formato insignia de Ares)

Más allá de usar juegos de terceros, Ares tendrá un **formato propio**: la **historia programable**. El niño **camina por una historia** (un mundo, una misión, un cuento), pero **para que los personajes actúen o respondan tiene que escribir el código**. El relato no avanza si el programa no es correcto: *programar es la única forma de pasar de página.*

Por qué es tan potente didácticamente:

- **Da sentido al concepto.** El bucle no es "repite 4 veces" en abstracto: es "el guardián repite su ronda hasta que el héroe se esconda". El personaje vuelve el concepto **concreto y con motivo**.
- **Construye el modelo mental de la máquina.** El personaje *es* la computadora obedeciendo literalmente. Si el niño escribe mal el orden, el personaje hace lo equivocado **y se ve** — eso enseña que la máquina no adivina, ejecuta. (Es el "notional machine" hecho personaje.)
- **La narrativa sostiene la práctica deliberada.** Querer saber *qué pasa después* mantiene al niño resolviendo reto tras reto sin sentir que "hace ejercicios".
- **Integra los tres hilos.** La historia puede tener arte/diseño (escenario), robotización (el personaje real es el robot) y programación (el código que lo mueve).

Anatomía de una historia programable (plantilla):

1. **Escena + misión:** se presenta el mundo y un personaje con un problema ("la abeja debe volver al panal").
2. **El personaje espera órdenes:** el niño escribe/arma el código (bloques o texto según banda).
3. **Corre → el personaje actúa:** retroalimentación inmediata y visible; si falla, la historia da una pista, no un castigo.
4. **El nuevo concepto es la única llave:** el capítulo no se puede ganar sin usar el bucle/condición/función que toca (el "filtro" de §4).
5. **Bifurcación:** las decisiones del niño (condicionales) cambian la historia → refuerza causa-efecto.
6. **Cierre del capítulo = "jefe":** un reto que combina lo aprendido; desbloquea insignia y el siguiente capítulo.

Cómo se construye en cada banda (mismo formato, sube la escalera de abstracción):

Banda	Cómo se "escribe el código" en la historia	Personaje / mundo
Exploradores	Bloques con íconos; tocar Makey Makey como "control"	Animalito/héroe en un mapa de cuadrícula
Constructores	Bloques (Scratch/Reeborg); decisiones ramifican el cuento	Explorador en un mundo con obstáculos y reglas
Inventores	Texto (MicroPython); el personaje del cuento **es el robot Pico**	Misión de rescate / laberinto físico
Innovadores	Texto pleno; varios personajes/sistemas que se comunican	Ciudad/IoT: agentes que reaccionan a datos

Implementación realista: se puede empezar **sin construir un motor**: una historia programable se arma hoy con **Reeborg's World** (mundos personalizados con narrativa), **Scratch** (cuento interactivo donde se programa al personaje) o **secuencias en code.org**. Más adelante, un **motor propio en HTML/JS** (un editor de código + un "escenario" que ejecuta lo que el niño escribe y anima al personaje) sería un activo diferenciador de Ares y vive en la plataforma Academy. *Recomendación: prototipar UNA historia-capítulo antes de invertir en motor.*

---

5. La ruta, banda por banda (scope & sequence)

Para cada banda: los **conceptos** que se introducen, la **definición-en-flujo** (cómo se dice la primera vez), el **nivel de abstracción**, el **juego ancla gratuito**, y el **hito** que gatea avanzar.

Banda 1 · Exploradores (Transición–2°, ~5–8 años)

Abstracción: desenchufado + bloques con íconos (sin lectura fluida). *Placa:* Makey Makey + ScratchJr/Scratch.

Concepto	Se dice así (en el flujo)	Juego ancla (gratis)	Hito
Instrucción / secuencia	"Una **instrucción** es una orden. Un programa es una **lista de órdenes en orden**."	ScratchJr, Lightbot Hour, code.org Curso A–C, Blockly Maze	Ordena 4–6 pasos para lograr una meta
Evento	"Un **evento** es *cuando pasa algo* (toco, aplaudo) → entonces ocurre una acción."	Scratch (al presionar bandera), Makey Makey	Hace que algo pase al tocar/ presionar
Bucle simple	"Un **bucle** es repetir. En vez de saltar 4 veces, digo **repite 4: salta**."	Lightbot, Blockly Maze	Usa "repite N" para acortar una secuencia
Depuración	"**Depurar** es encontrar y arreglar el error."	(todos)	Encuentra por qué su programa no llegó

Banda 2 · Constructores (3°–5°, ~8–10 años)

Abstracción: bloques (mBlock/MakeCode) + C++ "asomada" (leer texto, no escribirlo). *Placa:* Arduino (alterna: micro:bit).

Concepto	Se dice así	Juego ancla (gratis)	Hito
Condición	"Una **condición** es una pregunta de *sí/no*: ¿está oscuro? ¿el botón está pulsado?"	Reeborg (¿hay pared al frente?), code.org D–F	Formula condiciones verdadero/falso

Concepto	Se dice así	Juego ancla (gratis)	Hito
Condicional (si / si-no)	"Un **condicional** *decide*: **si** está oscuro **entonces** enciende; **si no**, apaga."	Reeborg, Scratch (juego con reglas)	Programa una decisión que cambia el comportamiento
Bucle condicional	"**Repite hasta que** llegues a la meta / **mientras** haya pared."	Reeborg, Blockly Maze avanzado	Repite con base en una condición, no en un número
Variable	"Una **variable** es una *cajita con nombre* que guarda y cambia un valor: el **puntaje**."	Scratch (hacer un juego con puntaje/vidas)	Crea y actualiza un puntaje en su juego
Operadores	"Comparo (> < =) y combino con **y** / **o** / **no**."	Scratch, Reeborg	Usa comparaciones y lógica en una condición

Banda 3 · Inventores (6°–8°, ~11–13 años)

Abstracción: puente **bloques** → **texto** (BIPES) → escribir MicroPython. *Placa:* Raspberry Pi Pico.

Concepto	Se dice así	Juego ancla (gratis)	Hito
Transición a texto	"El mismo bloque, ahora **escrito**: `for i in range(4):`."	Reeborg en Python, Code Club World, BIPES	Reescribe en texto un programa que hizo en bloques
Función	"Una **función** es un bloque con **nombre** que reutilizo: defino	CodeCombat (cap. gratis), CheckiO, Reeborg	Define y reutiliza una función

Concepto	Se dice así	Juego ancla (gratis)	Hito
	`avanzar()` una vez y la llamo muchas."		con parámetro
Lista / iterar	"Una **lista** guarda *muchos* valores con un nombre; un `for` los recorre."	CheckiO, Reeborg	Recorre una lista para resolver un reto
Descomposición	"Parto el problema en **funciones** pequeñas."	(proyecto Pico)	Divide un programa en 3+ funciones
Tipos de dato	"Un número, un texto, un `True/False`: cada uno se usa distinto."	CodinGame, CheckiO	Distingue y convierte tipos al leer un sensor

Banda 4 · Innovadores (9°–11°, ~14–17 años)

Abstracción: texto pleno, varios lenguajes. *Placa:* ESP32 + web (HTML/CSS/JS/SQL/dashboards).

Concepto	Se dice así	Juego ancla (gratis)	Hito
Estructuras de datos	"Un **diccionario** asocia clave→valor (sensor→lectura)."	CodinGame, CheckiO, Codewars	Modela datos del proyecto con dict/ listas anidadas
Clase / objeto	"Una **clase** junta *datos + acciones*: un objeto `Sensor` que sabe `leer()`."	CodinGame (bots), Ozaria/ CodeCombat	Crea una clase para un componente del sistema

Concepto	Se dice así	Juego ancla (gratis)	Hito
Eventos asíncronos / concurrencia	“Responder *mientras* pasan cosas: interrupción de sensor, mensaje de red.”	(proyecto ESP32/IoT)	Maneja un evento de red/sensor sin bloquear
Manejo de errores	“Si algo falla, lo **atrapo** y sigo (`try/except`).”	CheckiO, Codewars	Protege la lectura/red ante fallos
Estado y comunicación	“El sistema **recuerda** (base de datos SQL) y **habla** (WiFi/MQTT).”	(proyecto IoT + dashboard)	Guarda y muestra datos en un dashboard

6. Compuertas de maestría (cómo sabemos que aprendió)

Nadie avanza de banda por edad: avanza por **hitos demostrados** (o entra a refuerzo). Cada compuerta se evalúa con los 5 criterios de la rúbrica de convergencia (doc 04 / 19 §8): *descomposición · lógica · representación · depuración · autonomía*.

Compuerta	El estudiante debe ser capaz de…
Sale de Exploradores	Ordenar una secuencia con un bucle simple, hacer que algo pase con un evento y encontrar un error simple.
Sale de Constructores	Programar una **decisión** y un **bucle condicional**, usar una **variable** (puntaje) y representar su lógica en un diagrama de flujo.
Sale de Inventores	Pasar bloques↔texto, escribir y reutilizar una **función**, recorrer una **lista** y descomponer un programa.

Compuerta	El estudiante debe ser capaz de…
Sale de Innovadores	Modelar datos, usar una **clase**, manejar un **evento de red**/ **sensor** y errores, y conectar a un **dashboard**.

La evidencia vive en un **portafolio** (diagramas, pseudocódigo, capturas del juego y del código) — no en un examen de memoria.

---

7. Cómo se mantiene efectivo: el ciclo de cada sesión

El camino se recorre con el **ciclo de lección** del doc 20, anclado al juego. Una sesión típica:

1. **Juega** (desenchufado o el juego ancla): vive el concepto ganando un nivel.
2. **Predice y corre** (PRIMM): mira un programa que funciona, adivina qué hace, lo corre.
3. **Define en el flujo:** una frase + ejemplo + uso (nunca glosario).
4. **Representa:** dibuja el diagrama/pseudocódigo de lo que jugó.
5. **Modifica y crea:** cambia el nivel/juego, luego hace el suyo.
6. **Traza y depura:** sigue el programa a mano; arregla un bug.
7. **Aplica al "jefe":** lleva el concepto al proyecto del robot.

Repetido en **espiral** (más profundo cada banda) y con **práctica deliberada** (muchos retos cortos con respuesta inmediata) es lo que convierte el juego en habilidad transferible. **La depuración es un hilo explícito**, presente desde Exploradores (hay una trayectoria de aprendizaje de depuración propia en la investigación).

---

8. Mapa de juegos y herramientas (todos verificados, junio 2026)

Herramienta	Banda	Lenguaje	Costo	Nota
ScratchJr	Explor.	Bloques (íconos)	Gratis	Tablet; pre-lectores
Blockly Games	Explor.–Constr.	Bloques	Gratis, sin cuenta	Maze, Bird, Turtle
Code.org CS Fundamentals	Explor.–Constr.	Bloques	Gratis	Cursos A–F graduados
Lightbot (Hour of Code)	Explor.–Constr.	Bloques	Versión gratis	Secuencia y procedimientos
Scratch	Constr.+	Bloques	Gratis	Hacer juegos = eventos/variables
Reeborg's World	Constr.–Invent.	Bloques **y** Python	Gratis	Karel; bloques→texto, mismo reto
mBlock / MakeCode	Constr.	Bloques (+C++/Py)	Gratis	Liga con Arduino/micro:bit
BIPES	Invent.	Bloques→MicroPython	Gratis	Puente al Pico
Code Club World	Invent.	Bloques→Python	Gratis	Raspberry Pi Foundation
CodinGame	Invent.–Innov.	25+ lenguajes	Gratis	Juegos/bots con código real
CheckiO	Invent.–Innov.	Python/JS	Gratis (12+)	Puzzles como juego
Codewars	Innov.	Muchos	Gratis	Katas gamificadas
CodeCombat		Python/JS	**Freemium (~11**	

Herramienta	Banda	Lenguaje	Costo	Nota
	Invent.–Innov.		**niveles gratis)**	Solo degustación

9. Base de evidencia (honesta)

- **CSTA K-12 Computer Science Standards (2017):** bandas K-2 / 3-5 / 6-8 / 9-12 (= nuestras 4 bandas); progresión de bucles (repeat → repeat until → while), condicionales (if → if/else) y funciones.
- **K-8 Learning Trajectories (Rich, Strickland, Franklin et al., 2017):** trayectorias de *secuencia, repetición, condicionales, variables, descomposición y depuración* derivadas de +100 estudios; sustentan el orden de §3.
- **PRIMM (Sentance et al.)** y **Usar-Modificar-Crear (Lee et al.):** leer antes de escribir; sustentan la escalera de autonomía.
- **Aprendizaje basado en juego:** meta clara + retroalimentación inmediata + dificultad graduada; con el **filtro** de §4 para evitar *edutainment* sin transferencia.

Nota de rigor: la evidencia respalda el **orden de conceptos** y el **ciclo**, no afirma que un juego concreto "enseñe solo". El aprendizaje ocurre por el **puente** juego→representación→código→proyecto, guiado por el docente.

10. Pendiente (lo que sigue de este camino)

1. **Validar la ruta** contigo (educador): ¿el orden y los hitos calzan con lo que ves en aula?
2. Producir, con el doc 20, la **lección modelo** de un concepto (sugiero *el bucle*, Constructores) usando su juego ancla, de punta a punta.
3. Escalar a la **ruta completa de lecciones** banda por banda.
4. **Curar y probar** los niveles concretos de cada juego (qué nivel de Reeborg/Blockly entrena qué concepto) → un "set de misiones" por banda.

5. Montar el seguimiento de hitos y portafolio en la plataforma Academy.

Fin del documento v0.1. Es la columna vertebral; el doc 20 (método + plantilla) y los libros cuelgan de aquí. Se evalúa con la rúbrica de convergencia (04 / 19 §8).