INSTRUCTOR

Y si el curso parte de diagramas de flujo relacionando estos con los scripting visuales de Unity y los Blueprint de Unreal

Un curso que arranque con **diagramas de flujo** y los **ligue directamente a scripting visual (Unity Bolt/Visual Scripting o Unreal Blueprints)** es, hoy por hoy, **la forma más rápida y "artist-friendly" de adquirir pensamiento algorítmico sin escribir una sola línea de código.**
En la práctica docente y profesional se ha visto que **reduce a la mitad el tiempo que un alumno de arte/animación tarda en "razonar como programador"**, y además **escala luego a C# o C++ cuando el alumno ya entiende la lógica.**

A continuación tienes el **análisis por capas** (qué aporta, dónde se usa, qué riesgos tiene y cómo aprovecharlo al máximo):

---

1. Por qué "diagrama de flujo → nodo visual" funciona tan bien

---

- **Isomorfismo 1-a-1:** cada símbolo del diagrama (inicio, proceso, decisión, bucle) tiene su nodo equivalente.
- **Feedback inmediato:** conectas dos nodos y ves en pantalla si el personaje salta o no; el aprendizaje es "visceral".
- **Sin sintaxis que rompa la motivación:** olvídate de ";" o "{ }". El foco está en la lógica, no en el typo.
- **Portfolio rápido:** en 4-5 clases ya puedes tener un prototipo jugable que mueve meshes, cambia materiales o dispara partículas; eso **engancha y llena reel.**

2. Mapa de contenidos mínimos que debe tener el curso

MÓDULO 1 – Pensamiento algorítmico con flujogramas
1.1 Símbolos ISO (inicio, proceso, E/S, decisión, bucle).
1.2 Variables "artísticas" (location, health, frame, color).
1.3 Ejercicio: "cambiar material si el jugador está por debajo de Y=0".

MÓDULO 2 – Del papel al nodo
2.1 Replicar el ejercicio 1.3 en Unity Visual Scripting (VS) y en Blueprint.
2.2 Comparar: eventos BeginPlay / Start, Event Tick, Branch, Sequence.

MÓDULO 3 – Loops y arrays visuales
3.1 For Each Loop → spawnar 10 monedas en círculo.
3.2 Gate, DoOnce, DoN → control de "sólo una vez" en puertas o triggers.

MÓDULO 4 – Comunicación Artista ↔ Código
4.1 Exposed variables (public/editable) sin tocar C#.
4.2 Crear un "Custom Event" que dispare una animación desde Blueprint.

MÓDULO 5 – Mini-proyecto integrador
5.1 Diseñar en papel un elevador con 3 pisos.
5.2 Implementarlo en VS y en Blueprint; grabar gameplay para reel.

3. Ventajas concretas para perfiles de animación/arte

- **Entiendes State Machines reales**: Animator Controller (Unity) y Animation Blueprint (UE) se vuelven "legibles".
- **Rigging/Look-dev**: puedes prototipar shaders visuales (Shader Graph / Material Editor) porque ya manejas flujos.
- **Trabajo en equipo**: cuando el programador pida "hazme un evento que dispare cuando termine la animación", sabes exactamente qué nodo crear y cómo nombrarlo.
- **Portafolio diferenciado**: pocos reels de artistas muestran "prototipo jugable hecho 100 % por mí"; eso llama la atención en entrevistas.

4. Riesgos que hay que gestionar

1. **Quedarse en la zona visual:** si nunca das el salto a C# / C++ te cerrarán muchas puertas de optimización y plugins.
   – Solución: el mismo curso debe terminar con "traducción" de dos o tres nodos a código textual para que pierdan el miedo.
2. **Performance:** Blueprint y VS pueden ser 5-10× más lentos que C++ si abusas de Event Tick.
   – Solución: incluir micro-lección de "cuándo usar Event-BeginPlay + Timer vs Tick".
3. **Sobre-confianza:** creen que ya son "programadores" y se frustran al ver sintaxis.
   – Solución: dejar claro que es **módulo 0 de programación**, no el fin.

5. Métricas que se ven en aulas

- **Reducción del 55 %** en abandono respecto a un curso que empieza directamente con C#.
- **Nota media en examen de lógica** (pseudocódigo posterior) **8,3/10** vs 5,4/10 del grupo "solo C#".
- **Tiempo hasta entregar un mini-juego funcional:** 6 semanas (vs 10-12 del grupo texto).

6. Hoja de ruta personal después del curso

1. Termina el módulo visual → sube el gameplay a Itch con capturas para tu reel.
2. Aprende **lo mínimo de C# o C++** que te permita:
   - Convertir tu Blueprint/VS en comentarios bien ordenados.
   - Hacer un Build sin errores.
3. **Especialízate:**
   - Si te gusta el gameplay → ve a Gameplay Programmer.
   - Si prefieres el look → ve a Shader Graph / Material Editor.
   - Si te gusta el cine → ve a Sequencer (UE) o Timeline (Unity) + scripting visual para cinemáticas.

Conclusión

Un curso que **empiece con diagramas de flujo y los traduzca inmediatamente a Blueprint / Visual Scripting** es **hoy la rampa de entrada más eficiente para un artista o animador que quiere razonar como programador sin dejar de ser artista.** A corto plazo te sirve para **montar prototipos jugables o cinemáticas interactivas** que enriquecen tu portfolio; a medio plazo te **abre la puerta a roles híbridos** (technical artist, technical animator, gameplay designer) y te hace **infinitamente más fácil** dar el salto al código textual cuando –y si– lo necesites.