

Desarrollo de Videojuegos

**¿Como empezar en el
Desarrollo de Videojuegos?**

Licencia Creative Commons

Reconocimiento-NoComercial-SinObraDerivada 2.5



Usted es libre de:

- copiar, distribuir y comunicar públicamente la obra.

Bajo las condiciones siguientes:



Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador.



No comercial. No puede utilizar esta obra para fines comerciales.



Sin obras derivadas. No se puede alterar, transformar o generar una obra derivada a partir de esta obra.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor

Contenidos

Introducción	4
Herramientas necesarias	6
Conocimientos necesarios	8
APIs Gráficas	11
Estructura básica de un videojuego	13
Recursos	17
APIs Gráficas y complementarias	17
IDEs/Compiladores	18
Libros	19
Gráficos	19
Programación	19
Inteligencia Artificial	19
Matemáticas y Física	20
Scripting	20
Networking	20
Sonido/Música	20

Introducción

La mayoría de las personas que han estudiado o están cursando alguna carrera relacionada con computación e informática y han pasado por ramos relacionados con la programación, se han preguntado alguna vez: *¿Cómo se hace un videojuego?*, *¿Qué tengo que saber?* *¿Por donde empiezo?* y muchas preguntas más. Muchos han intentado hacerlo sin resultado alguno, otros en cambio han podido crear sus primeros juegos con un poco de esfuerzo, pero aun así siempre quedan preguntas pendientes por responder y bastante por aprender.

Este artículo pretende responder algunas de esas preguntas y dar a conocer que entrar al mundo del desarrollo de videojuegos, no es algo tan complicado, pero se necesita siempre estar investigando, estudiando, leyendo sobre el tema, y lo más importante practicar mucho, ya se sabe que la práctica hace al maestro.

Existe gran cantidad de libros relacionados con el desarrollo de videojuegos, la gran mayoría se encuentra en idioma inglés, quizás para los que están empezando esto sea un impedimento, así que ya podemos contestar una de las preguntas *¿Qué tengo que saber?*, debemos tener por lo menos algun conocimiento básico de ingles para entender documentos, artículos, libros, etc. para estar al día con la tecnología que se aplica en el mundo de los videojuegos. Al final de este artículo podremos ver un listado de algunos libros que son útiles para comenzar con el desarrollo de videojuegos.

Cuando se habla de desarrollo de juegos, se suele pensar que existe un programador, que debe ser capaz de hacer todo el juego, pero pensar esto es un gran error, ya que realizar un videojuego involucra aparte de programadores, otros personajes como diseñadores, músicos, escritores, etc. Este artículo se centrará en los programadores, pero hay que tener en cuenta que si se quiere producir un buen juego, además de la originalidad, debe tener buenos gráficos, buena música, jugabilidad, etc. Aunque todo lo que involucra este proceso puede ser construido por un solo programador, es más adecuado tener a un especialista en cada área.



Para comenzar en el desarrollo de videojuegos es preferible hacerlo con proyectos pequeños, con el fin de ser capaces de cumplir las metas que nos fijamos, nunca pensar en proyectos extremadamente grandes que se nos escapen de las manos, ya que lo mas probable es que nunca los terminemos, lo cual nos llevará a la frustración.



Es común que las personas que se dedican a esta área del desarrollo, comiencen con remakes de juegos como: Pong, Tetris, Arkanoid, Pacman, Spice Invaders, Sokoban, Snake, Mario Bross, etc. Es un muy buen ejercicio desarrollar algun juego antiguo con algunas modificaciones, aprenderemos bastante. Se pueden realizar videojuegos muy buenos y entretenidos basándonos en las mejores características de otros, solo basta dejar correr nuestra imaginación.

La selección de juegos nombrados anteriormente son algunos de los primeros que aparecieron en la historia de los videojuegos, como el famoso Pong. Todos estos juegos tienen algo en común, son juegos en 2D. Hoy en día la gran mayoría de los títulos que salen al mercado son juegos 3D como Doom, Quake, Call of Duty, Unreal Tournament, Need for Speed y un largo etcétera.

Muchos pretenderán comenzar a desarrollar un juego del tipo Doom, tienen que pensar que realizar uno de estos juegos de forma independiente (solo un programador) es una tarea titánica, además todos estos videojuegos fueron realizados por grandes equipos de personas (no solo programadores) y con presupuestos millonarios. Esto no quiere decir que no podamos crear uno de estos, pero hay que ir paso a paso. Cuando ya tengamos dominado los juegos 2D, pasaremos a crear algún proyecto 3D de un tamaño moderado o una buena idea es adaptar uno de nuestros juegos 2D a 3D.



Unreal Tournament 2004



Need for Speed Most Wanted

Herramientas necesarias

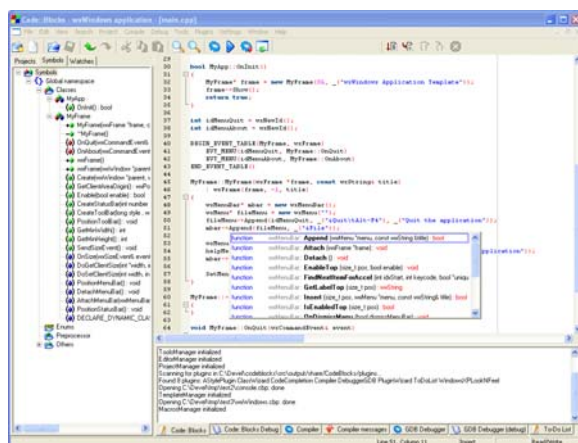
Ahora veremos que herramientas son necesarias para convertirse en un desarrollador de videojuegos. Antes que todo necesitamos manejar algún lenguaje de programación, entre mejor sea el nivel de conocimiento que tengamos de aquel lenguaje más rápido avanzaremos.

El lenguaje por excelencia usado en la historia del desarrollo de los videojuegos profesionales ha sido **C** y desde la última década el gran Dios es **C++**, debido a las ventajas que este posee, ya que hace uso del paradigma de la programación orientada a objetos, aunque también podemos usar características de la programación procedural de C.

Pero no solo basta con conocer un lenguaje de programación para que nuestras ideas de lo que queremos que realice el juego sean escritas en este lenguaje, sino que también necesitamos “algo” que nos transforme todo ese código que solo entienden los humanos a otro código que entienden las máquinas, estamos hablando del famoso **compilador** que realiza esta conversión a ceros y unos. Existen compiladores para cada tipo de lenguaje y estos pueden ser gratuitos o comerciales.

Ya dijimos que la mayoría de los proyectos son realizados en C/C++ pero hay otras alternativas, existen lenguajes compilados e interpretados. Ejemplo de un lenguaje compilado es C/C++, ejemplo de lenguajes interpretados (en vez de un compilador necesitan de un interprete para funcionar) son Python, Basic, etc.

Como ejemplo de compilador tenemos a **MinGW** (Minimalist GNU for Windows) el cual es un port del **GCC** de Linux. Este compilador es gratuito, pero no solo es el compilador, sino que también contiene otras herramientas que nos apoyan en el proceso de desarrollo, como es el linker, make, debugger, profiler, etc.



Además del lenguaje y el compilador, necesitamos un **editor de texto** donde podamos escribir nuestras líneas de código, es aquí donde aparecen los Entornos de Desarrollo Integrado mas conocido como **IDE** (Integrated Development Environment). IDEs hay para todos los gustos, pero existe una bastante cómoda de usar y con muy buenas características, estamos hablando de **Code::Blocks**. También puede ser usado **Dev-C++** el cual ya trae en su distribución el compilador GCC, aunque desde hace mucho

tiempo no es actualizado. Code::Blocks tiene varias características que lo hace superior a Dev-C++, una de ellas es el visor de funciones, variables, clases y la completación de

código, que todo programador agradecerá. Otro Compilador/IDE bastante usado en estos tiempos es **Microsoft Visual C++**.

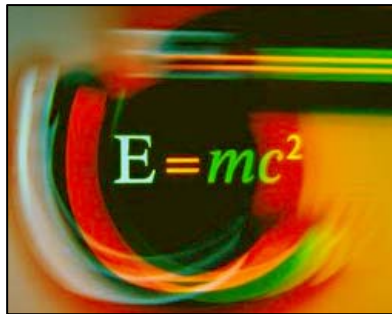
Conocimientos necesarios

Ya conocemos cuales son las herramientas básicas para comenzar, pero esto es solo la punta del iceberg, necesitamos otros conocimientos que probablemente en un principio para juegos sencillos no sean necesarios, pero lo serán a medida que el juego vaya aumentando su complejidad.

Para comenzar conviene saber sobre el manejo de **estructuras de datos**, esto es parte de los conocimientos básicos que debería tener cualquier programador.

Hoy en día disponemos de un conjunto de bibliotecas que nos facilitan el uso de estructuras de datos en nuestros programas, estamos hablando de la biblioteca **STL** (Standard Template Library), la cual ya es un standard de C++, y lo podemos encontrar en todos los compiladores actuales.

Para no entrar en detalle sobre esta biblioteca (lo cual debe ser discutido en otro artículo) solo diremos que STL es una colección de estructuras como listas, pilas, colas, vectores, etc. y algoritmos que manejan estas estructuras de forma transparente al programador. Conviene aprender a usarlas ya que nos ahorrarán mucho trabajo y nuestro código quedará más ordenado y elegante.



Siguiendo con los conocimientos, también debemos saber **matemáticas**, no tenemos que ser unos genios en esta área, pero ayuda bastante tener conocimientos de trigonometría y algebra lineal. Para comenzar no es tan necesario saber todo esto, pero mientras los proyectos se van haciendo más complejos, será necesario comenzar a revisar los viejos cuadernos y libros de matemáticas que tenemos guardados.

Si hay que saber matematicas, no podíamos dejar afuera la **física**, pilar fundamental para ciertos juegos que simulan micromundos, donde intervienen fuerzas gravitacionales, velocidades, aceleración, masas, en general cualquier variable física. Básicamente en juegos de tipo plataformas como Mario Bross, se aplican ciertas cosas de cinemática básica en los personajes, como en el movimiento que realizan, cuando saltan, caen, etc. y por supuesto todo el tema relacionado con las colisiones y respuesta es parte de la física del juego. Existe una API que nos facilita el manejo de la física, se llama **ODE** (Open Dynamics Engine) la cual es Open Source. En un principio para construir nuestros primeros videojuegos la física será muy sencilla (si es que la tiene), y no será necesario hacer uso de complicadas bibliotecas.

Podemos hablar también del **networking**, que es todo lo relacionado con la comunicación de datos a través de redes (acá aparecen términos como sockets, TCP,

UDP, etc.). Por ejemplo en todo juego multijugador debemos estar enviando datos de posiciones, puntajes y otras características de personajes, objetos, etc. a otras máquinas para que exista una interacción entre todos los jugadores con el fin de ver reflejado en nuestro PC lo que están haciendo los otros jugadores en sus propios computadores. Esto es usado hoy en día en todos los juegos multijugador, gran ejemplo de esto es Unreal Tournament. En esta área también disponemos de bibliotecas especializadas en el tema, un ejemplo es SDL_Net.



No podemos dejar afuera el **scripting**, que abarca todo lo relacionado con la generación de archivos *scripts*, donde cada uno de estos contiene variables, datos y código. Por ejemplo si en un juego necesitamos crear interfaces, o mantener posiciones de ciertos objetos, es más cómodo tener un archivo con esta información y que desde el programa principal se lea el archivo y se interprete, esto facilita mucho la modificación del juego, tanto en el proceso de desarrollo como en futuras actualizaciones, y nos permite liberarnos de estar compilando cada vez que hagamos un cambio en el código fuente. Existen lenguajes de scripting que nos facilitan la vida como Python y LUA.

También debemos aprender **Inteligencia Artificial**, ¿Qué sería de un juego donde los enemigos se comportan de forma poco “inteligente”?, simplemente no sería divertido jugar, y rápidamente aprenderíamos los movimientos y su comportamiento. Aquí es donde podemos aplicar conocimientos de IA en nuestros juegos, dándole un toque más real y mas entretenido.



Algunos métodos de la IA aplicadas al desarrollo de juegos son el PathFinding (Implementado a través del Algoritmo A* para la búsqueda del camino más corto entre dos puntos) aplicado en juegos como Starcraft, cuando por ejemplo queremos mover un personaje a cierto punto del mapa y este es capaz de llegar a ese punto esquivando todos los objetos que encuentre a su paso y usando el camino más corto. Otros métodos aplicados son los algoritmos genéticos, vida artificial, máquinas de estados finitos, árboles de búsqueda, también áreas extensas de la IA como las redes neuronales, etc.

Mediante scripts podemos generar el comportamiento que tendrá un personaje en un juego, a estos personajes se les llama NPC (Non Playing Character).

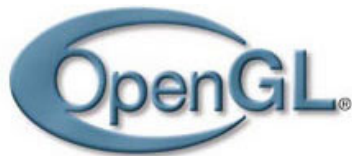


Starcraft

APIs Gráficas

Antes de terminar con los conocimientos no podemos dejar de hablar de algo importantísimo, el manejo de una API Gráfica. Una **API** (Application Programming Interface) o Interfaz de Programación de Aplicaciones, es un conjunto de funciones que realizan tareas específicas facilitando la vida al programador. Al hablar de API grafica nos referimos a un set de funciones para inicializar por ejemplo modos gráficos, realizar copiado de datos de la memoria del computador a la tarjeta de video (blitting), etc. Existen APIs específicas para cada tipo de tarea que queramos realizar.

Actualmente las APIs Gráficas más usadas son OpenGL y DirectX.



OpenGL (Open Graphics Library) es una API multiplataforma creada por Silicon Graphics en 1992, que maneja solo el aspecto gráfico de un sistema, dejando afuera el sonido, música, control de teclado, mouse, joysticks, gamepads, etc., los que deben ser controlados con otras APIs especializadas. Existe una biblioteca multiplataforma para el manejo de audio tridimensional llamada OpenAL (Open Audio Library).

DirectX es una API multimedia creada por Microsoft en 1995, que consta básicamente de Direct3D para la parte gráfica, DirectSound y DirectMusic para la parte de audio, y DirectInput para el control de teclados, joysticks, etc. También incluye DirectPlay para comunicación de datos en redes.



El uso de estas dos APIs en un comienzo puede ser un poco complicado y engorroso, especialmente DirectX. Existen algunas alternativas como Allegro o SDL, las que son multiplataforma.



SDL (Simple DirectMedia Layer) es una API gráfica para realizar operaciones de dibujado en 2D, gestionar efectos de sonido y música, y cargar imágenes. Existen varias librerías que complementan SDL, algunas de ellas son:

SDL_image: Carga de imágenes en diferentes formatos: png, jpg, etc.

SDL_mixer: Carga de formatos de sonido como wav, mp3, ogg, etc.

SDL_net: Comunicación de datos en redes.

SDL_ttf: Uso de fuentes TrueType.

SDL_gfx: Dibujo de primitivas gráficas, escalar/rotar imágenes, control de framerate, Filtros de imagen MMX.

SDL permite básicamente crear aplicaciones en 2D, pero si queremos extender estas capacidades podemos usarla en conjunto con OpenGL, ya sea para crear nuevamente aplicaciones 2D que aprovechen las características de aceleración por hardware, que todas las tarjetas de video poseen hoy en día, como para crear aplicaciones completamente en 3D.

Ya vimos las herramientas y conocimientos necesarios para comenzar en el desarrollo de videojuegos, pero no podemos dejar de mencionar que también existen los **Engines** o **Motores** de juego, que están basados en algunas de las APIs ya vistas, OpenGL o DirectX principalmente, que proveen al programador todas las funcionalidades necesarias para el desarrollo de un juego.

Básicamente un Engine esta formado por varios sistemas y subsistemas, por ejemplo un sistema gráfico para manejar objetos 2D o 3D, un sistema de control de entrada (teclado, mouse, etc.), sistema de texto, sistema de red, sistema de scripts, sistema de audio, etc. Todos los conocimientos necesarios que vimos antes ahora están aplicados en un Engine.

Existen Engines gratuitos (algunos bastante buenos y complejos) y otros comerciales, por ejemplo podemos nombrar los siguientes: Irrlicht, Ogre 3D, Crystal Space 3D, PopCap, Torque, etc.

Para comenzar a desarrollar conviene utilizar algo más sencillo (pero no por eso menos potente). Por ejemplo podríamos usar la dupla *SDL/OpenGL* junto a otras librerías externas para construir nuestro propio Engine.

No solo es necesario disponer de las herramientas y conocimientos que ya vimos para el desarrollo de juegos, aun hay más, pero esto ya no es tarea del programador (aunque conviene también tener estos conocimientos en forma general por lo menos) estamos hablando del **Diseño Gráfico** de un juego. Alguien debe ser capaz de realizar las imágenes que aparecerán, ya sea en menús, pantallas de configuración y en el juego mismo.

Los típicos conocimientos que debe tener un diseñador grafico, son el uso de alguna herramienta de retoque fotográfico como Photoshop, Paint Shop Pro o Gimp, y herramientas para modelar objetos 3D tales como 3D Studio Max, Maya, Blender, entre otros.

Y no puede quedar afuera tampoco, algun personaje que componga la música o cree y/o edite los sonidos que aparecerán en el juego. Ejemplo de este tipo de software es Adobe Audition, Acid Pro, Reason, Fruity Loops, etc.

Estructura básica de un videojuego

Antes de terminar veremos básicamente como funciona por dentro un videojuego.

Primero hay que aclarar que un videojuego es un programa diferente a los programas convencionales.

Un videojuego debe funcionar en tiempo real, en todo momento mientras se está ejecutando, el juego debe estar realizando alguna tarea como: dibujar los objetos, actualizar coordenadas, calcular colisiones, etc., independiente de si el usuario hace algo. Obviamente debe también estar esperando que ocurra algún evento, ver si el usuario presiona alguna tecla, si mueve el mouse, o presiona algún botón de este y luego actuar en consecuencia. Todo esto y más ocurre en un ciclo o loop.

Básicamente la estructura de un videojuego consta de las siguientes partes:

- 1. Inicialización**
- 2. Ciclo del videojuego**
 - 2.1 Entrada**
 - 2.2 Procesamiento**
 - 2.3 Salida**
- 3. Finalización**

Ahora veamos una explicación en más detalle de cada una de ellas:

1. Inicialización: aquí inicializaremos todo lo que será usado luego en el ciclo del videojuego. Por ejemplo aquí inicializaremos la librería gráfica, un modo gráfico, el sistema de sonido/música, de texto y cualquier otro tipo de sistema necesario. Además reservaremos memoria para los objetos que intervienen en el juego, creación de estructuras de datos, etc. Carga de sonidos, de imágenes y de recursos en general. También en este proceso se inicializarán las posiciones iniciales de los personajes, carga de puntajes desde un archivo, etc.

2. Ciclo del videojuego: el ciclo del videojuego es un loop que se estará repitiendo una y otra vez. Aquí es donde ocurre toda la acción del juego, y la única forma para poder salir de este ciclo es cuando el jugador pierde, llega al final del juego o sale del videojuego con alguna combinación de teclas o presionando algún botón del mouse, etc. El ciclo del juego consta básicamente de tres partes:

2.1. Entrada: en esta parte se obtiene desde algún dispositivo de entrada (teclado, mouse, joystick, etc.) todo lo que realiza el jugador, por ejemplo que tecla presionó/soltó del teclado, que botón del mouse presionó/soltó, si movió el mouse en alguna dirección, etc.

2.2. Procesamiento: aquí se procesa toda la información que se recibió en el punto anterior y se toman decisiones a partir de los datos de entrada. Es decir aquí está toda la lógica del juego. Se procesa la física, inteligencia artificial, comunicación de datos en red, etc.

2.3. Salida: en este punto se muestra toda la información ya procesada en el punto anterior, aquí es donde mostramos los gráficos en pantalla, reproducimos sonidos, etc.

3. Finalización: por último en esta parte se hace básicamente lo opuesto a lo que hicimos en la inicialización, es decir, eliminar de la memoria todos los recursos almacenados, ya sea imágenes, sonidos, música, etc. Cerrar todos los sistemas que se abrieron en la inicialización. Guardar datos de puntajes en un archivo, etc.

Si quisiéramos ver lo anterior en lenguaje C, la estructura básica se vería así:

```
int main(int argc, char **argv)
{
    int salir=0;

    inicializacion();

    while(!salir)
    {
        entrada();
        procesamiento();
        salida();
    }

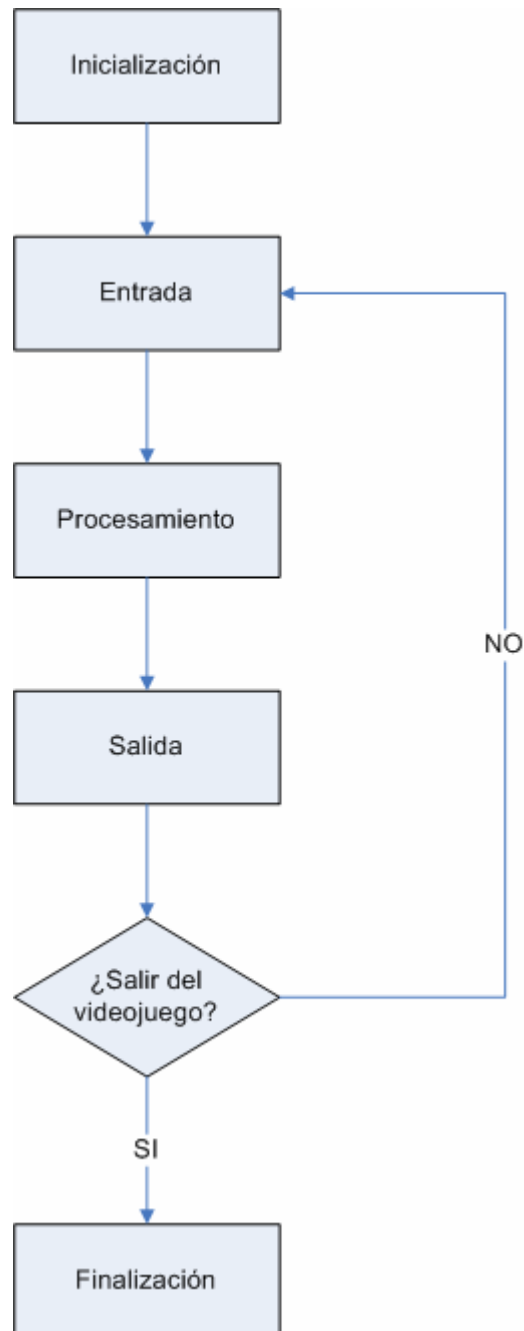
    finalizacion();

    return 0;
}
```

Resumiendo vimos que los conocimientos básicos para desarrollar juegos son:

- Lenguaje de programación.
- Compilador/IDE.
- Matemáticas.
- Física.
- Networking.
- Scripting.
- Inteligencia Artificial.
- API Gráfica.
- API para Sonido y Música.

Y la estructura básica de un videojuego la podemos ver en un simple diagrama de flujo:



Recursos

APIs Gráficas y complementarias

- **SDL**
<http://www.libsdl.org>
- **SDL_image**
http://www.libsdl.org/projects/SDL_image/
- **SDL_mixer**
http://www.libsdl.org/projects/SDL_mixer/
- **SDL_ttf**
http://www.libsdl.org/projects/SDL_ttf/
- **SDL_net**
http://www.libsdl.org/projects/SDL_net/
- **SDL_gfx**
<http://www.ferzkopp.net/joomla/content/view/19/14/>
- **OpenGL**
<http://www.opengl.org>
- **OpenAL**
<http://www.openal.org>
- **DirectX**
<http://www.microsoft.com/windows/directx/>

IDEs/Compiladores

- **Code::Blocks**
<http://www.codeblocks.org>
- **Dev-C++**
<http://www.bloodshed.net/dev/>
- **Visual C++ Express 2005**
<http://www.microsoft.com/spanish/msdn/vstudio/express/VC/>
- **Mingw32**
<http://www.mingw.org/>

Libros

Aquí tenemos algunos libros que nos ayudarán a adquirir más conocimientos y reforzar los que ya tenemos sobre el desarrollo de videojuegos. Todos estos ejemplares pueden ser adquiridos por medio de [Amazon](#).

Gráficos

- **Focus On SDL**
- **OpenGL(R) Reference Manual: The Official Reference Document to OpenGL, Version 1.4**
- **OpenGL Game Development**
- **OpenGL Game Programming**
- **OpenGL SuperBible**
- **Tricks of the Windows Game Programming Gurus**
- **Game Programming All in One**
- **3D Game Programming All in One**

Programación

- **C++ How to Program**
- **Data Structures for Game Programmers**
- **Core Techniques and Algorithms in Game Programming**

Inteligencia Artificial

- **AI for Game Developers**
- **AI Techniques for Game Programming**
- **Programming Game AI by Example**
- **AI Game Engine Programming**

Matemáticas y Física

- **Mathematics and Physics for Programmers**
- **Physics for Game Programmers**
- **Fundamentals of Math and Physics for Game Programmers**

Scripting

- **Game Scripting Mastery**
- **Programming in Lua**
- **Game Development With Lua**
- **Game Programming with Python, Lua, and Ruby**

Networking

- **Networking and Online Games: Understanding and Engineering Multiplayer Internet Games**
- **Algorithms and Networking for Computer Games**

Sonido/Música

- **Creating Music and Sound for Games**
- **Audio Programming for Interactive Games**