ÍNDICE

INTRODUCCIÓN

En los años que llevo enseñando desarrollo de videojuegos me he topado con una increíble cantidad de alumnos que piensan que no tienen la capacidad de aprender a programar, piensan que es algo muy complejo, que necesitas un amplio conocimiento lógico y matemático, antes de aprender incluso lo más simple tienen una actitud errónea hacia aprender algo nuevo, ese es probablemente el error más común y que más aleja a los principiantes que se están iniciando en el mundo de la programación, tengo muy claramente identificada esa actitud y esa sensación porque yo mismo estuve ahí; para mí la programación era algo en extremo complejo y me sentía abrumado con tan solo pensarlo, durante mucho tiempo esa era la idea que tenía en mi cabeza y fue el primer gran obstáculo que yo mismo me puse antes de siquiera intentarlo, lo que no entendía es de dónde venía esa concepción, porque yo al igual que mucha gente creemos que aprender a escribir código es algo sumamente complicado de hacer, si bien es cierto que en el campo se puede llegar a escribir código bastante complejo debido a la necesidad de resolver problemas igualmente complejos; la realidad es que no se empieza por algo de esas características, incluso profesionales que dedican su vida a crear código nunca en su vida laboral llegan a escribir algo que requiera de una complejidad muy alta y mucho menos empezaron a aprender con algo semejante.

Tiempo después me di cuenta de que cuando yo era un estudiante de universidad hace casi veinte años los recursos con los que contaba eran presentados de una manera poco clara para la mayoría de los estudiantes, desde libros inmensos con explicaciones por concepto y estructuras cuadradas que no fomentaban el esfuerzo por el autoaprendizaje y el razonamiento sobre los problemas presentados hasta profesores que solo lograban que el alumno contestara correctamente lo que el programa escolar requería, cosa que de por sí estaba bastante alejada de los requerimientos de la industria para la cual se suponía te estaban preparando. Hoy en día, así como podemos encontrar recursos no muy buenos y engañosos, también podemos hallar recursos increíblemente buenos, ya sea en libros, escuelas o en internet; cada vez hay más y mejores lugares para aprender o mejorar tus

habilidades de programación o desarrollo de videojuegos, lamentablemente estas opciones en idioma español aún son bastante limitadas, de este problema surge la necesidad de crear esta obra con el afán principal de que el lector tenga un punto de partida amigable , sólido y suficiente para seguir adelante por su cuenta siendo capaz de discernir entre un buen lugar para seguir incrementando sus capacidades y otro que no lo es tanto.

Aprender a programar no es algo difícil, es algo que requiere paciencia y dedicación, en estos días donde se busca tanto una gratificación inmediata se vuelve muy sencillo desistir en el intento y concluir que simplemente es algo muy complicado para ti cuando en realidad no lo es. Te pido que tengas esto en mente mientras navegas por las páginas de este libro o de cualquier otro recurso que utilices para incrementar tus capacidades; muy pronto el código que estarás escribiendo empezará a cobrar sentido y cuando esto suceda te prometo que tendrás una sensación de satisfación tan gratificante que te impulsará a seguir adelante, solo te pido la disciplina de dedicarle un poco de tiempo todos los días, de hacerlo, en pocas semanas esa idea de que la programación es algo complejo habrá desaparecido y serás completamente libre de disfrutar cada vez que aprendes algo nuevo.

Todos aprendemos a un ritmo diferente, si mientras estás aprendiendo algo en este libro te sientes un poco abrumado o desanimado porque no estás entendiendo algo, no te preocupes, despéjate un poco y luego regresa a donde te quedaste; si te parece que sigue siendo complicado es muy probable que algo del conocimiento previo adquirido en este libro no haya quedado del todo claro, en ese momento puedes regresar un paso, o dos, o tres y practicar un poco lo que ya entendiste para que esas bases te permitan entender de forma más clara lo que viene más adelante. No te sientas presionado por terminar este libro, el tiempo que pasarás aquí deberá de ser divertido, además de servirte para aprender muchísimas cosas nuevas.

No existe una forma definida de desarrollar un videojuego, se pueden crear dos videojuegos prácticamente idénticos utilizando métodos completamente diferentes, en este libro aprenderás a utilizar Unity, el motor de videojuegos más popular del planeta, y también aprenderás a programar en C#, que es un lenguaje de programación que le pertenece a Microsoft pero que podemos utilizar para crear comportamientos dentro de Unity, vamos a trabajar en un ambiente muy específico y aprenderemos una o dos formas de hacer las mismas cosas. Esto no quiere decir que son las únicas formas de hacer algo pero puedes estar seguro de que son formas sólidas de lograr algo, últimamente he trabajado en conjunto con el departamento de educación de Unity Technologies para identificar cómo tener buenas bases a la hora de aprender a desarrollar en Unity y esas bases están aplicadas en este libro, espero que lo disfrutes.

1 FUNDAMENTOS DE TECNOLOGÍA Y COMPUTACIÓN

¿Qué es y cómo funciona una computadora?

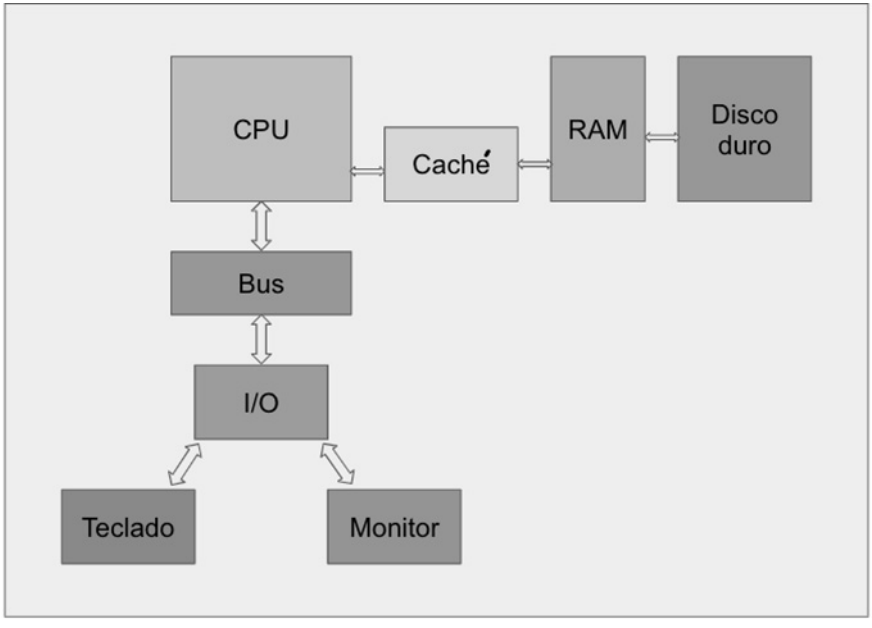
Muchas veces aprendemos a utilizar algo sin realmente comprender cómo funciona y en computación esto es cada vez más común, estamos acostumbrados a utilizar la computadora, nuestro smartphone, televisiones y tecnología en general sin darnos cuenta de lo que realmente está pasando con nuestro dispositivo, por lo tanto, me parece relevante que antes de aprender a programar hablemos brevemente sobre algunos puntos clave sobre tecnología y computación.

Antes que de cualquier otra cosa quiero hablar sobre qué es una computadora, prácticamente todo el mundo las utilizamos a diario y aunque no es necesario entenderlas en esencia para hacerlas funcionar, sí es importante hacerlo para sacarles el mejor provecho, pero entonces, ¿qué es una computadora? Bueno, en esencia es un dispositivo capaz de ejecutar una serie de operaciones; así de simple, estas operaciones pueden ser lógicas y aritméticas, ¿y de qué forma le podemos indicar qué operaciones va a realizar esa computadora?, tal vez lo adivinaste, escribiendo código de programación.

Basados en esta definición de lo que es una computadora podemos entender que una computadora no es solo el objeto que comúnmente llamamos computadora, hoy en día nuestros teléfonos, televisiones, relojes, electrodomésticos y demás tecnología que utilizamos día a día también son computadoras, aunque de un uso más específico todos en su mayoría disponen de un CPU, el encargado de ejecutar estas operaciones.

Y bueno, ya que empecé a hablar sobre el CPU, este chip es el corazón de cada computadora, hay uno (o varios) dentro de tu computadora, tu smartphone o tu televisión, también los encontramos en nuestros automóviles, aviones, impresoras, equipos de sonido, etc. Es también muy común que un CPU se ayude de otros chips como los de comunicación vía Wi-Fi o tecnologías de comunicación celular, de unos años hasta aquí se ha vuelto muy popular el internet de las cosas que se basa en que prácticamente todas las cosas estén conectadas a internet, por ejemplo, hoy en día es muy común tener en casa cosas muy comunes como cafeteras, tostadores o hasta la iluminación conectadas a internet.

Hablando de chips auxiliares para el CPU, creo que es buen momento de analizar cómo se compone la arquitectura básica de una computadora, obviamente no todas ellas son iguales, pero mostraré una arquitectura que podría considerarse como base para la mayoría de las arquitecturas, tal vez has escuchado hablar sobre memoria, discos duros o dispositivos de entrada y salida, bueno para demostrar cómo están conectados veamos el siguiente diagrama.



Aunque en este diagrama se muestra como una vista general de una arquitectura cabe mencionar que no es exacto ni hace referencia a alguna arquitectura en particular, sin embargo, funciona para demostrar el flujo de datos ya sea en su lectura o escritura.

Si hablamos de dispositivos de entrada como un teclado o dispositivos de salida como un monitor, ambos son controlados por una interfaz de entradas y salidas *I/O*

(Input y Output) que utiliza un *Bus* de datos para comunicarse con el CPU, existen diferentes tipos de *Buses* y para verlos de forma sencilla podemos verlos como carreteras que transportan los datos dentro de nuestra computadora.

La parte de entrada y salida de una computadora resulta interesante, sin embargo, como programadores nos interesa más la parte de memoria y almacenamiento, ya hablamos de lo que es y cómo funciona un CPU, pero ¿de dónde vienen los datos y comandos que el CPU ejecuta? Si vemos el diagrama nos daremos cuenta de que existen tres tipos de memoria en nuestra arquitectura, la memoria caché, la memoria RAM y el disco duro, pero ¿para qué necesitamos tres? Cada uno de estos tres niveles de almacenamiento tiene sus propias características, la memoria caché es la más rápida pero también la más pequeña, es la más cercana al CPU y por lo tanto toma mucho menos tiempo llevar información de esta memoria a procesar dentro de nuestro CPU, un procesador moderno comúnmente tiene 12MB (Megabytes) o más de esta memoria, hablaré un poco sobre los MB más adelante, también es la memoria más cara y eso es una de las razones principales por las cuales los fabricantes no pueden poner mucha de esta memoria, y bueno, obviamente no podemos trabajar con esa cantidad de memoria, tan solo una imagen en alta resolución tomada con una cámara moderna ocupa más espacio que lo que tenemos disponible de memoria caché, es por eso que podemos hacer uso de la memoria RAM que en computadoras actuales es común encontrarla en cantidades de 16GB (también hablaré de los Gigabytes más adelante) dependiendo de las necesidades del usuario este número puede ser menor o mayor, usualmente cuando mencionamos cuánta memoria tiene una computadora nos estamos refiriendo a la memoria RAM, es mucho más barata de fabricar que la memoria caché y podemos tener mucho más de ella, es obviamente más lenta que la memoria caché aunque en los últimos años las velocidades en este tipo de memoria se han incrementado considerablemente, el recorrido que los datos hacen desde la memoria RAM al CPU es considerablemente más grande que desde la memoria caché, además en casi cualquier arquitectura de computadora tenemos disponible una memoria o dispositivo de almacenamiento masivo, en este caso estamos hablando de un disco duro, generalmente podemos ver configuraciones de computadoras con 1TB (Terabytes) o más utilizando dispositivos magnéticos, sin embargo, la popularidad de tecnologías como los discos de estado sólido ha ido en aumento y aunque su costo es considerablemente mayor son preferibles porque la diferencia en rendimiento es muy grande, aun así, no han logrado ser tan rápidos como la memoria RAM además de que el recorrido de los datos es aún mayor, y bueno, creo que podemos decir que la necesidad de estos tres tipos de memoria se debe mayormente a costos de producción.

Al hablar de memoria me gustaría opinar un poco sobre unidades de almacenamiento, empezando por el **bit** que es la unidad mínima de almacenamiento

en computación y tal vez lo más importante que debemos saber del bit es que solo puede almacenar un número a la vez, y algo muy interesante es que este número solo puede ser un 1 o un 0. Probablemente has escuchado de un sistema de lógica binaria en la que todas sus variables y operaciones lógicas están basadas en combinaciones del número 1 y el número 0, ahora, si en un bit solo cabe uno de estos dos números, ¿cómo puedo hacer combinaciones? Aquí es donde podemos hacer uso de un **byte** un byte también es una unidad de almacenamiento pero que puede contener hasta ocho bits al mismo tiempo, de hecho en computación no le podemos mandar bits individuales a un CPU para que haga su procesamiento, la unidad mínima que podemos utilizar para hacerlo es un byte y en este caso estaríamos hablando de un CPU de ocho bits, si, como el del **NES** (o Nintendo viejito como dicen algunos), chips más modernos no suelen utilizar ocho bits, después de pasar por los dieciséis y treinta y dos, hoy nos topamos comúnmente con procesadores de sesenta y cuatro bits lo que quiere decir que cada instrucción que le llega a este CPU se encuentra en un paquete conformado por sesenta y cuatro bits, a este paquete también se le conoce como **palabra**, es decir, un conjunto finito de bits que el CPU puede manejar a la vez.

Comenté que iba a hablar sobre Megabytes, Gigabytes y Terabytes, bueno, si entendiste correctamente el párrafo anterior será muy sencillo darte cuenta de que estas también son unidades de almacenamiento, aquí dejo una tabla que me parece deja más claro la relación entre estas unidades.

Bit	Puede tener el valor 1 o el valor 0
Byte	Conjunto de 8 bits
Kilobyte	1000 bytes
Megabyte	1000,000 bytes
Gigabytes	1000,000,000 bytes
Terabyte	1000,000,000,000 bytes

Las computadoras solo entienden el lenguaje usando bits y empaquetado en bytes, cuando nosotros hacemos código para crear un programa lo que estamos haciendo es comunicarnos con la computadora; sin embargo, sería muy complicado y lento estar hablando el mismo lenguaje que utiliza la computadora, esta es la principal razón de que existan los lenguajes de programación, en esencia un lenguaje de programación es una abstracción del lenguaje máquina para hacerlo mucho más

entendible para los seres humanos y que nos permite comunicarnos con una computadora de manera mucho más sencilla; no todos los lenguajes de programación tienen el mismo propósito ni funcionan igual, hay muchos lenguajes de programación en la industria , sin embargo, son pocos los que son realmente populares alrededor del mundo, como ya debes saberlo en este libro nos enfocaremos en utilizar y comprender **C#** que es uno de los diez lenguajes de programación más populares que existen y es extremadamente versátil ya que se puede utilizar para hacer cosas como simples programas de consola, páginas web, manejo de bases de datos y videojuegos entre otras cosas.

C# y .Net

En este capítulo voy a hablar un poco de qué es C#, qué es .NET, las diferencias entre ellos y por qué los programadores principiantes comúnmente confunden los términos.

Empezaré diciendo que C# (*pronunciado C Sharp*) es un lenguaje de programación y que .NET es un *framework*.

Un lenguaje de programación es utilizado para expresar un conjunto de instrucciones para una computadora, luego esta computadora las ejecuta en forma de otro lenguaje conocido como lenguaje máquina, existen muchos lenguajes de programación, pero son unos pocos los que son realmente populares alrededor del mundo, C# es uno de ellos y podemos utilizarlo para gran variedad de aplicaciones como desarrollo para dispositivos móviles, desarrollo web y desarrollo de videojuegos.

Un *framework* es un conjunto de módulos de software estructurados que comúnmente sirven de base para nuevos desarrollos, podemos decir que es código previamente creado y estructurado del cual podemos hacer uso dentro de nuestro software y que permite potenciar y agilizar nuestro desarrollo, *.NET framework* está compuesto de dos componentes, Class Library que es básicamente una serie de librerías de código que podemos utilizar en nuestro código y CLR (Common Language Runtime) del cual hablaremos con más detalle en el siguiente apartado.

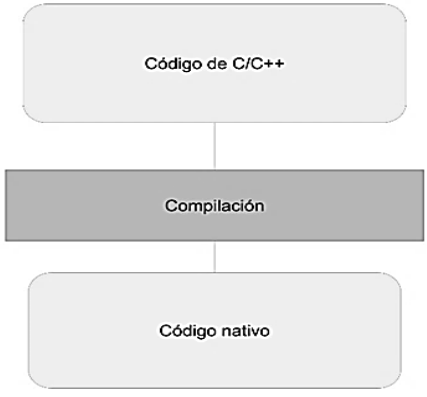
Entonces podemos concluir que C# es el lenguaje en el cual escribimos nuestras líneas de código, generamos nuestras estructuras y algoritmos para crear los comportamientos que buscamos y además tiene acceso a .NET que es un *framework* del cual nos podemos ayudar para utilizar comportamientos específicos predefinidos o hacer que nuestro software funcione en diferentes sistemas operativos sin necesidad de volver a escribir nuestro código.

Es importante mencionar que el acceso a .NET no es exclusivo de C#, existen otros lenguajes de programación que pueden hacer uso de este *framework*.

CLR

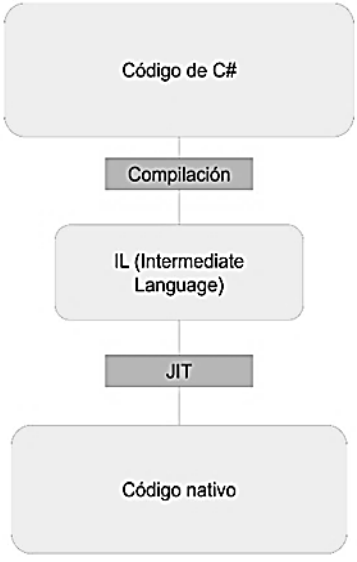
Antes de explicar en concreto lo que es CLR hablaré un poco sobre la historia de C#, este lenguaje pertenece a la familia de los lenguajes conocidos como tipo C, debido a que hereda muchas de sus propiedades del lenguaje de programación C creado por **Dennis Ritchie,** disponible desde 1972 y que hasta hoy es ampliamente utilizado, el lenguaje C es imperativo y de uso general, otro integrante de esta familia es el lenguaje de programación C++, otro lenguaje de uso general que además de poder ser imperativo también puede ser orientado a objetos (tema del siguiente capítulo) y genérico, C++ fue diseñado por **Bjarne Stroustrup** y fue visto por primera vez en 1985, también es un lenguaje ampliamente utilizado en la actualidad sobre todo porque sigue siendo constantemente actualizado, hablando de C#, este fue diseñado por Microsoft y apareció por primera vez en el año 2000, podríamos decir que C++ y C son respectivamente el papá y el abuelo de C#, al igual que C++, C# es un lenguaje orientado a objetos, pero su principal y gran diferencia radica precisamente en CLR.

En el caso de C/C++ podemos escribir nuestro código y luego compilarlo, esto va a resultar en código nativo, podemos representarlo con esta pequeña imagen.



En la parte de arriba está el código escrito en C/C++, luego pasa por el proceso de compilación y obtenemos código nativo, al hablar de código nativo hacemos referencia a código creado específicamente para una plataforma o sistema operativo, por ejemplo podemos escribir código en C/C++ que tenga como objetivo el sistema operativo Windows, entonces el código nativo resultante solo se ejecutará en este

sistema, podríamos escribir otra versión de nuestro código para Linux y otra para Mac OS, pero es precisamente aquí donde empiezan los problemas de los lenguajes C y C++, en cambio C# gracias al uso de CLR nos permite escribir nuestro código una sola vez y hacerlo funcionar en más de una plataforma sin mucho esfuerzo por parte del desarrollador; ¿cómo funciona esto? Bueno, a diferencia de sus ancestros, C# no compila hacia código nativo, en cambio lo hace a un lenguaje llamado IL (Intermediate Language) y aquí es donde entra al juego CLR, CLR es una pequeña aplicación en memoria que toma el código intermedio o IL y lo convierte a código nativo, a este proceso se le conoce como JIT (Just-in-time Compilation), aquí vemos el proceso representado gráficamente.



Entonces mientras la plataforma que estemos utilizando tenga instalado CLR será capaz de ejecutar nuestro software creado con C# y que solo escribimos una sola vez.

Programación orientada a objetos

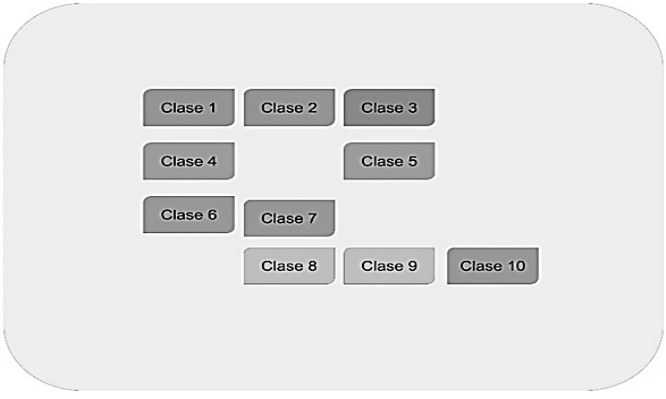
La programación orientada a objetos nos permite escribir código como si fueran objetos de la vida real, está basada en el concepto de objetos que contienen datos en forma de campos conocidos como atributos y procedimientos en forma de funciones o métodos. Una característica del software que está escrito orientado a objetos es que todo el código está dentro de objetos y estos objetos interactúan entre sí. C# es

un lenguaje orientado a objetos que está basado en clases, lo que significa que los objetos son instancias de esas clases, podemos pensar en una clase como los planos para construir una casa y la casa en sí sería un objeto, es decir, una instancia de la clase.

Los lenguajes orientados a objetos como C# tienen muchas más características, podríamos adentrarnos en temas como la herencia de clases, el polimorfismo y la encapsulación pero en este punto solo causarían confusión dado que esas características son temas de nivel intermedio o avanzado y tratar de verlos antes de siquiera haber escrito nuestras primeras líneas de código te va a causar frustración innecesaria dado que serían mucho más complejos de entender si aún no tienes bien fundamentadas las bases, así que no te preocupes por eso, eventualmente entenderás esos temas y es importante entender que se pueden hacer programas completos y complejos con solo utilizar código básico, incluso podrías hacer un videojuego entero solamente sabiendo las bases de programación siempre cuando estas estén sólidamente fundamentadas para poder hacer un producto de calidad.

Arquitectura de una aplicación .NET

Como vimos en el apartado anterior, C# es un lenguaje orientado a objetos basado en clases, entonces todo el código que vamos a escribir será parte de una clase y probablemente tendremos varias clases comunicándose entre sí, lo podemos representar así.



Lo que quiero representar en esta imagen es que podemos tener en nuestro proyecto varias clases, algunas pueden estar relacionadas entre sí, otras no, sin embargo, todo debe de estar contenido en una de ellas, esta podemos decir que es la vista general de nuestro proyecto, pero ¿qué hay dentro de cada una de estas clases? Bueno, ya lo vimos un poco en el apartado de programación orientada a objetos, las clases están compuestas de dos cosas, procedimientos y atributos, es decir, que si