**PROGRAMACIÓN PARA LA CREACIÓN DE VIDEOJUEGOS**

Los programas se escriben por sentencias (frases de código) y en la mayoría de sus partes corren de la sentencia de más arriba a la sentencia de más abajo (ya veremos cómo esto puede cambiar). Un ejemplo sería algo así:

Computador, suma 3 + 4 -> Esto sería una sentencia

Computador, escribe el resultado de la suma en un archivo de texto -> Esto sería otra sentencia, que ocurriría luego de la sentencia de la suma.

Todo programa está hecho de 2 cosas fundamentales: Variables y Funciones. Estos conceptos son los que revisaremos primero, pues son los más importantes. Luego, veremos que son los operadores, las sentencias de control y los objetos.

Todos los conceptos de programación que se cubrirán a continuación se aplican en su totalidad a la creación de videojuegos.

**Variables**

Pueden ser imaginadas como cajones que guardan información dentro. Para que cualquier programa pueda operar tiene que trabajar sobre algún tipo de información. Si no fuera así, sería como pedirle a un escultor que hiciera una escultura con aire. Las variables son la materia sobre la que trabaja un programa.

**Declaración de una variable**

Para crear una variable son necesarias dos cosas como mínimo: un tipo de dato y un nombre. Recordemos que las variables actúan como cajones: el tipo de dato es el tipo de cosa que ese cajón guarda dentro y el nombre es el nombre que le damos al cajón. Siempre terminamos la declaración de una variable con un punto y coma. Cabe hacer notar que las variables pueden guardar sólo un tipo de dato dentro de ellas. Un cajón de calcetines solo puede guardar calcetines.

Ejemplos de declaración de variables:

int nivelActual;

float distanciaAEnemigo;

bool muerto;

string nombreJugador;

Tipos de variables

Los tipos de variables más comunes son los siguientes:

int: Número entero, positivo o negativo. Sus valores se escriben como simples números (3, 458029, -25).

float: Número de punto flotante (con decimales), positivo o negativo. Sus valores se escriben como números con los decimales tras un punto y con una ‘f’ al final (28.4f, -0.3f, 12f).

bool: No es un número, es una de dos cosas: true o false (verdadero o falso). Una variable que guarda un bool suele representar el estado binario de algo (muerto o vivo, en el aire o en el piso, invulnerable o vulnerable, etc). Sus valores son true o false, y se escriben tal cual.

string: Conjunto de letras (o caracteres). Un string puede guardar cualquier cantidad de letras, símbolos y números. Se pueden utilizar por ejemplo para guardar nombres de personajes o diálogos. Sus valores se escriben como cualquier combinación de caracteres, pero con comillas al principio y al final (“I am Error”, “Player 1”, “No pasar”, “Hola aventurero, soy Gruk el adivino y vengo a dejarte claro que los strings pueden ser muy largos”).

**Asignarle un valor a una variable**

Para guardar un valor concreto dentro de un cajón (variable) se utiliza el símbolo ‘=’:

int vida = 15;

float distancia = 13.5f;

bool saltando = false;

string nombreJugador = “John”;

No es necesario asignarle a una variable un valor en su declaración. Puedo declarar una variable y asignarle su valor luego. Por ejemplo:

int vida;

\*más código\*

vida = 10;

**Funciones**

Si las variables son piezas de lego, las funciones son las instrucciones para armar cosas con ellas. Las funciones, también llamadas métodos, representan la lógica de un programa, lo que ese programa hace.

Una función encapsula un algoritmo: una serie de pasos lógicos, un método a seguir.

En un programa podemos encontrar una función de dos formas: la declaración de la función y la invocación de la misma función.

La declaración de una función es donde se escriben los pasos lógicos que esa función representa.

La invocación son partes del programa donde se le pide al computador que ejecute la función en cuestión, es decir, que realice los pasos lógicos indicados en su declaración.

Una función también puede ser entendida como una máquina que realiza alguna actividad. Esa máquina puede recibir y/o retornar información. Un ejemplo sería una máquina dispensadora de bebidas. Para usar esta máquina debo darle información de entrada (que tipo de bebida quiero sacar). Luego de eso la máquina ejecuta una serie de procesos. Finalmente la máquina me retorna un resultado (la bebida que yo escogí).

**Declaración de una Función**

Dada nuestra metáfora de la máquina de bebidas (ver párrafo anterior), si yo quiero declarar una función tengo que indicarle al menos 3 cosas:

* Qué datos necesita para funcionar (el tipo de bebida en nuestro ejemplo).
* Qué debe hacer con esos datos.
* Qué tipo de dato nos entregará como resultado la función.

Analicemos la siguiente función:

int Sumar (int a, int b)

{

int resultado = a + b;

return resultado;

}

Lo primero que escribimos en este caso fue ‘int’. Esta primera palabra debe indicar el tipo de dato que la función dará como resultado. En este caso la función retornará un tipo de dato int (número entero).

Luego del tipo de dato de retorno escribimos el nombre que le daremos a la función. En este caso la función se llama Sumar. Este nombre puede ser cualquier cosa que queramos. Es importante darle un nombre para poder referirnos a esta función en otros lugares de código cuando queramos utilizarla.

Tras declarar el nombre escribimos entre paréntesis qué tipos de datos recibe la función como entrada (sería la selección de bebida en el ejemplo anterior). Necesitamos separar los datos que la función recibe con una coma. A cada dato de entrada debemos también darle un nombre. En el caso de la función Sumar se indica que para poder ejecutarla es necesario pasarle dos datos enteros. La razón por la que hay que darle un nombre a cada dato (en este caso a y b) es para poder referirnos a esos datos dentro de la función. Fíjate que dentro de la función dice “a + b”, ¿cómo podríamos referirnos a esa suma si hubiéramos puesto en la declaración “(int, int)”?

Los datos de entrada se conocen como parámetros.

Ahora toca ver qué hacer con los datos que la función recibió como entrada. En el caso de nuestra función Sumar, lo que hago es crear una variable de tipo entero llamada resultado y guardar dentro de esa variable (recuerda que es como un cajón) el valor dado por la expresión a + b (ya veremos cómo funcionan las expresiones como a + b).

Finalmente debemos retornar un dato (darle la bebida a nuestro usuario). Para retornar un dato simplemente escribimos return seguido del dato que retornaremos (o del cajón que contiene ese dato).

Cabe hacer notar que una función puede no retornar ningún dato y puede no requerir ningún parámetro. Un ejemplo de estas dos cosas a la vez sería esta función:

void FuncionSinParametrosQueNoRetornaNada ()

{

\* la función hace algo acá \*

}

Las funciones que no retornan valores (void) no necesitan tener escrito “return”.

**Llamada a una función**

Lo que hemos visto de funciones hasta este punto es la declaración de una función. Ahora que ya sabemos qué es lo que hace, podemos hacer que lo haga. Cuando le pedimos a una función que ejecute el proceso que escribimos en su definición (declaración) estamos llamando a esa función.

Analicemos la siguiente llamada a la función Sumar que escribimos recién:

Sumar (3, 4);

Acá estoy llamando a la función Sumar. Fíjate que en su declaración me pide dos enteros a los que llamamos a y b. En este punto yo le estoy diciendo cuál será el valor que tomará a y b. a y b funcionan como variables, son cajones que pueden guardar datos. Lo que estoy haciendo en esta llamada a la función es decir: a guardará el valor 3 y b guardará el valor 4.

En el caso de una llamada a una función no escribimos un par de llaves luego de escribir el nombre de la función y los parámetros, simplemente terminamos la sentencia con un punto y coma.

Cuando el programa ejecuta esta sentencia, todo lo que se había escrito en la declaración de la función Suma se ejecutará. Copio la declaración de la función para facilitar el análisis de lo que ocurre:

int Sumar (int a, int b)

{

int resultado = a + b;

return resultado;

}

Cuando escribimos Sumar (3, 4); Los pasos de lo que ocurre son los siguientes:

int Sumar (int a = 3, int b = 4) -> Se le asignan valores a los parámetros

{

int resultado = a + b;

return resultado;

}

int Sumar (int a = 3, int b = 4)

{

int resultado = 3 + 4; -> Las referencias a los parámetros se reemplazan por estos valores

return resultado;

}

int Sumar (int a = 3, int b = 4)

{

int resultado = 7; -> Se guarda 7 en el cajón llamado “resultado”

return resultado;

}

int Sumar (int a = 3, int b = 4)

{

int resultado = 7;

return 7; -> Reemplazamos la referencia a “resultado” con su valor y retornamos 7.

}

Llego a la última llave y la función termina.

Los nombres de los parámetros y variables siempre se reemplazan por el valor que contienen cuando llega el momento de ejecutar una sentencia, excepto cuando se escriben a la izquierda de un signo ‘=’; En este caso teníamos: resultado = a + b, que se transforma durante su ejecución en resultado = 3 + 4. En este caso a y b, que estaban a la derecha del signo ‘=’ fueron reemplazados por los valores que guardaban.

Entonces, cuando escribo Sumar (3, 4) todo lo descrito ocurre.

**Operadores**

Los operadores son signos que se ponen entremedio de dos valores y que operan sobre ellos para darnos un resultado. Un típico operador con el que todos estamos familiarizados es el signo ‘+’. Cuando escribo en mi código ‘3 + 6’, es lo mismo que haber escrito 9.

Una cosa que nos puede ayudar a juntar las piezas que hemos visto es saber que los operadores son funciones que se escriben de un modo distinto (literalmente son eso). Por ejemplo cuando escribo ‘a \* b’ internamente se llama a una función que multiplica dos valores que recibe como parámetro. Sería algo así como ‘Multiply (a, b)’.

**Operadores más comunes**

+ y - : Los operadores de suma y resta, todos sabemos como funcionan.

\* y / : Los operadores de multiplicación y división, respectivamente.

% : El operador de módulo. Cuando escribo 4%3 lo que recibo como resultado es el resto de la división 4/3, que en este caso es 1, porque 3 cabe una vez en 4 y sobra 1. Si escribo 3%4 el resultado es 3, porque 4 cabe 0 veces en 3 y sobran 3.

++ y -- : Operadores de incremento y decremento. Suman o restan 1 a un valor numérico. Si escribimos 3++ el resultado es 4, y si escribimos 3-- el resultado es 2. Se utilizan en la práctica sobre variables, no sobre valores numéricos como ‘3’ directamente. Por ejemplo podríamos encontrar la expresión ‘contador++’ que le sumaría 1 al valor que tenía actualmente la variable ‘contador’.

&& : Se llama “Logical AND” (Y lógico). Es un operador de valores booleanos, lo que quiere decir que no puedo escribir 3 && 5, pero sí puedo escribir true && false, false && false, true && true, etc. Da como resultado true si ambos valores son true y false si al menos uno de los valores es false. Generalmente cuando escribimos este operador no escribimos algo como true && false (porque sabemos lo que dará como resultado) sinó que usamos variables que guardan booleanos y que por lo tanto no podemos predecir lo que dará la operación cuando se ejecute. Un ejemplo de esto sería una sentencia como: vivo && soyOrco. No te preocupes demasiado en entender esto ahora mismo, porque será más fácil verlo en el contexto de un caso real en la sección de sentencias de control.

|| : Se llama “Logical OR” (O lógico). Es un operador de valores booleanos. Da como resultado true si alguno de los valores sobre los que opera es true, y false si ambos son falsos. Ver la descripción del operador && más arriba si no has leído antes sobre operadores booleanos. Ejemplos de su uso serían sentencias como: estoyEnPiso || tengoDobleSalto. No te preocupes si no lo entiendes bien en este punto, se entenderá en contexto en la sección de sentencias de control más adelante.

>, <, >= y <= : ‘Mayor’, ‘menor’, ‘mayor o igual’ y ‘menor o igual’, respectivamente. Son especiales en el sentido de que operan sobre valores numéricos pero dan un resultado booleano. 3 > 9 es falso porque 9 no es menor que 3. 3 >= 3 es verdadero porque 3 si es igual a 3. Si no has usado estos símbolos antes, recomiendo el siguiente truco: El valor que esté en el lado más pequeño del signo es el valor que estamos viendo si es menor que el otro. Lo mismo, el valor que esté en el lado más amplio del signo es el valor que estamos revisando si es mayor que el otro. Se entenderán mejor en el contexto de las sentencias de control más adelante.

== y != : ‘Igual’ y ‘distinto’, respectivamente. Pueden operar sobre cualquier valor, numérico, booleano, string (letras) y otros. Comparan si los valores a ambos lados del signo son iguales o distintos, y retornan el booleano correspondiente. 3 == 3 es verdadero, “hola” != “hola” es falso, porque “hola” y “hola” son iguales, no distintos. Ver sentencias de control más adelante para revisar estos operadores en contexto.

**Sentencias de control**

Las sentencias de control nos permiten controlar el flujo de un programa. Nos permiten, por ejemplo, ejecutar un pedazo de código u otro dependiendo de alguna condición.

**If**

El if se escribe de la siguiente forma:

if (condicion)

{

Haz lo siguiente

}

Lo que va dentro de paréntesis debe ser una expresión booleana, es decir cualquier cosa que se resuelva como un verdadero o un falso. Las llaves {} separan lo que se ejecutará si la sentencia es cierta del resto del código. Todo lo que esté entre las llaves se ejcutará sólo si la sentencia booleana es verdadera.

Analicemos un ejemplo concreto de esto:

if (vida <= 0)

{

EjecutarAnimacionDeMorir();

}

Dentro del if escribimos una operación booleana, en este caso el operador es <= (menor o igual, ver más arriba). Si el valor que contiene la variable vida es menor o igual a 0, el resultado de la operación será verdadero. Si la vida es mayor que 0, el resultado de la operación será falso.

Si la operación resulta verdadera entonces llamaremos a la función EjecutarAnimacionDeMorir, que podemos asumir que hace lo necesario para mostrar a un personaje muriendo en la pantalla.

**else**

La sentencia else se ejecuta en conjunto con la sentencia if. Juntas tienen la siguiente estructura:

if (condicion)

{

FuncionA();

}

else

{

FuncionB();

}

Esto significa que si se cumple la condición se ejecutará la FuncionA, y si no se cumple se ejecutará la FuncionB.

**else if**

La sentencia ‘else if’ puede usarse también en conjunto con la sentencia if. Juntas tienen la siguiente estructura:

if (condicionA)

{

FuncionA();

}

else if (condicionB)

{

FuncionB();

}

Esto significa que si se cumple la condicionA se ejecutará la FuncionA. Si la condicionA NO se cumple, entonces recién se revisará si se cumple la condicionB. Si esta se cumple se ejecutará la FuncionB, si NO se cumple no se ejecutará ninguna de las dos funciones.

**if, else if y else juntas**

El if, el else if y el else pueden usarse todas juntas en una expresión como la siguiente:

if (condicionA)

{

FuncionA();

}

else if (condicionB)

{

FuncionB();

}

else

{

FuncionC();

}

El resultado de esto es que si se cumple la condicionA se ejecuta FuncionA, si se cumple la condicionB se ejecuta la FuncionB y si no se cumple ninguna de las anteriores se ejecuta la FuncionC.

**while**

La sentencia while se encuentra en expresiones como la siguiente:

while (condicion)

{

Funcion();

}

Funciona de la siguiente manera: Primero vemos si la condición es cierta. De serlo ejecutamos Funcion. Luego, y esta es la parte que la diferencia del if, vemos NUEVAMENTE si la condicion es cierta, y si lo es ejecutamos nuevamente Funcion. Repetimos esto hasta que la condicion sea falsa. Cuando finalmente la condicion sea falsa continuamos el programa abajo del while normalmente.

Un ejemplo más práctico del uso del while se encontraría en una sentencia como esta:

while (viendoJugador)

{

AvanzarHaciaJugador();

}

Vemos si el enemigo ve al jugador, y de ser así avanzamos hacia él. Luego volvemos a preguntar si estamos viendo al jugador y repetimos. El resultado es que nos acercamos al jugador mientras lo tengamos a la vista.

**for**

Finalmente tenemos la sentencia de control for. Se encuentra en expresiones como la siguiente:

for (int contador = 0; contador < 10; contador++)

{

EscribirEnPantalla (contador);

}

En orden, la manera en que se ejecuta esta expresión es como sigue:

1 - Primero se crea una variable de tipo entero (int) llamada ‘contador’ y se inicializa con un valor de 0.

2 - Luego se revisa la condición ‘contador < 10’.

3 - Si esta condición es cierta se ejecuta la función EscribirEnPantalla (contador). Dado que contador guarda el valor de 0 actualmente, se escribiría 0 en la pantalla.

4 - Tras ejecutar la función se le suma 1 al contador (La expresion ‘++’ se explica más arriba en la sección operadores).

5 - Tras aumentar el valor de la variable contador se vuelve a preguntar si contador es menor que 10. En este caso contador vale 1 tras haberlo incrementado así que esta expresión sigue siendo cierta.

6 - Se vuelve a ejecutar EscribirEnPantalla, esta vez contador vale 1 así que se escribe ‘1’ en la pantalla.

7 - Se vuelve a incrementar contador en 1 y se vuelve a revisar la condición. Este proceso se repite hasta que la condición se vuelve falsa. En ese punto continuamos el programa debajo de la sentencia if (después de la llave ‘}’ que cierra el if).

**Clases y Objetos**

**Nota:** Si no hemos visto esta materia aún en clases **no** les recomiendo estudiarlo a partir de este documento. Las clases y objetos son un tema difícil del que surgen dudas en cada paso de su aprendizaje. La intención de este material es reforzar y aclarar puntos vistos en clases.

Siempre que escribimos una función o una variable lo hacemos dentro de una clase que la encapsula. Las clases son conjuntos de variables y funciones.

Una clase por sí misma es simplemente una definición. Es como el plano de un edificio. Se puede utilizar para crear edificios con ella (objetos). Imaginemos que tenemos en las manos el plano de un edificio que tiene un jacuzzi, y queremos sumergirnos en él. Tenemos que construir el edificio antes. El jacuzzi, y todas las cosas que el plano define, son las variables y funciones escritas en esa clase. Para poder utilizarlas, debemos crear objetos a partir de esa clase.

Una última metáfora antes de irnos a las definiciones más técnicas: Podemos crear edificios de distintos colores a partir del mismo plano, y el jacuzzi de un edificio puede tener más o menos burbujas que el jacuzzi del otro. Esto quiere decir que las variables que obtienen los objetos creados a partir de una clase pueden tener distintos valores.

La implementación más simple de una clase tiene la siguiente estructura

class Personaje

{

}

Con eso ya tengo una clase, un plano, del que puedo crear objetos.

Para crear un objeto de esta clase escribo, por ejemplo en la función Main:

void Main ()

{

Personaje p = new Personaje();

}

Lo que estamos haciendo acá es creando un nuevo objeto de tipo Personaje usando la expresión “new Personaje()”. Escribir new, seguido de la llamada a una función con el nombre de la clase, como en este caso, es la manera de crear objetos. Eso es lo que hacemos para construir el edificio, siguiendo el ejemplo anterior.

Esta función especial, “new Personaje()” retorna algo que podemos guardar en una variable. Retorna en particular un Personaje. Esto es importante, así como hay funciones que retornan “int”, que guardamos en variables de tipo “int”, podemos crear variables de tipo Personaje. La clase Personaje que escribimos **es un tipo de dato**, igual que int o bool.

La función new Personaje, utilizada para crear objetos a partir de la clase, tiene el apropiado nombre de **constructor**.

En estos dos trozos de código hemos definido una clase y creado objetos a partir de ella. El código que se muestra a continuación expande sobre lo mismo.

Esta sería la definición de la clase:

class Personaje

{

int vida;

public Personaje (int vidaInicial)

{

vida = vidaInicial;

}

}

Y esta sería la creación de objetos de esa clase:

void Main ()

{

Personaje jugador = new Personaje(10);

Personaje enemigo = new Personaje(5);

}

Lo que hicimos recién es añadirle a la clase Personaje una variable llamada vida y un constructor explícitamente. Los constructores se escriben de la forma descrita arriba, como una función con el nombre de la clase y sin valor de retorno.

Dado que hemos creado un constructor explícitamente, al momento de crear objetos de tipo Personaje, como en la función Main, **debemos** utilizar ese constructor. Fijémonos en que esta vez debemos darle un parámetro al constructor. El constructor asigna la vida inicial del objeto con el valor del parámetro entregado.

A esto nos referimos cuando decimos que los objetos creados a partir de una misma clase pueden tener sus diferencias. En este caso el objeto guardado en la variable jugador tiene 10 de vida y el objeto guardado en la variable enemigo tiene solo 5. Lo que comparten estos dos objetos es que ambos tienen vida, pues vienen de la misma clase, pero cuanta vida tienen puede variar de un objeto a otro.

Es por lo anterior que no puedo escribir Personaje.vida, como hemos visto (o veremos) en las primeras clases. Personaje es la clase, no tiene vida por sí misma. Lo que sí puedo hacer es escribir jugador.vida y enemigo.vida.