# Lección 2

En esta lección examinaremos la forma como se colocan imágenes 2D en una pantalla de un Windows Game y posteriormente aplicaremos algunas técnicas para mover la imagen por la pantalla, lo cual nos ayudara a entender la forma más básica como se animan los personajes en un juego.

**Paso 1: Iniciar un proyecto nuevo:** Inicia Microsoft XNA y ve a la opción de crear un nuevo proyecto (File -> New Project) y selecciona un proyecto de tipo Windows Game, puedes colocarle el nombre que quieras y seleccionar la ruta donde se va a guardar el proyecto a tu gusto (figura 1)

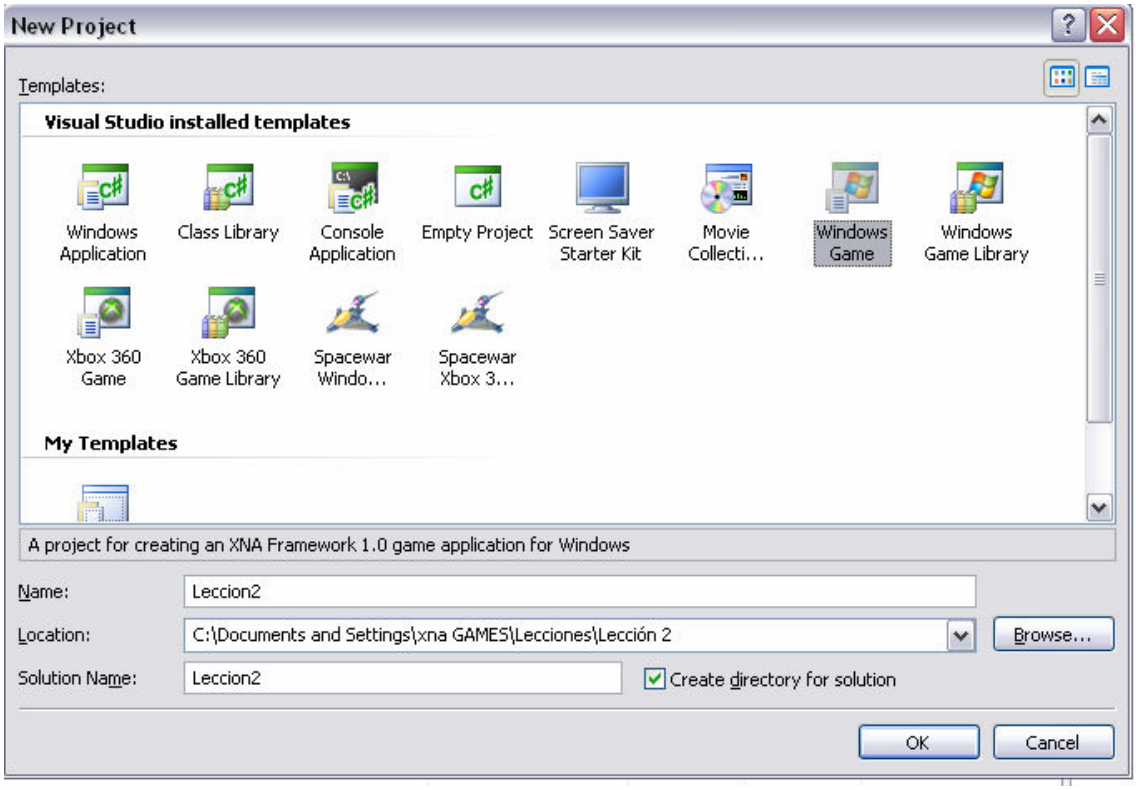


*Figura 1*
*Crear un nuevo proyecto*

Hecho esto, ya estamos listos para comenzar a crear nuestro juego. Antes de comenzar puedes probar ejecutando el juego tal como está, haciendo clic en el botón de ***debuggin*** ( ) con lo que obtendrás la característica ventana azul que vimos en la lección 1.

**Paso 2: Agregar una imagen:** A continuación seleccionaremos la imagen con la que vamos a trabajar en nuestro proyecto (puede ser cualquiera), yo seleccione la siguiente (figura 2), es recomendable para el propósito de este ejercicio, usar una imagen de un tamaño relativamente pequeño, por ahora es obligatorio usar una imagen con extensión .PNG, las cuales incorporan

transparencias (esto para que no se vea el marco que rodea la imagen), y para que XNA la reconozca

*Figura 2*
*Selecciona una imagen para trabajar (formato .png)*

1. Cuando tengas tu imagen ve a la carpeta que seleccionaste para guardar tu proyecto, y guarda la imagen allí, para ello abre un explorador de windows y navega por tu disco duro hasta el lugar donde Visual estudio guarda los proyectos, por defecto es:

***<usuario>\Mis documentos\Visual Studio 2005\Projects\<proyecto>***

2. Hecho esto vamos a incluir la imagen como un elemento de proyecto de nuestro juego. Para hacer esto, regresa a Visual Estudio y en explorador de soluciones, hacemos clic en el nombre del proyecto y seleccionamos la opción de agregar un elemento existente (figura 3).

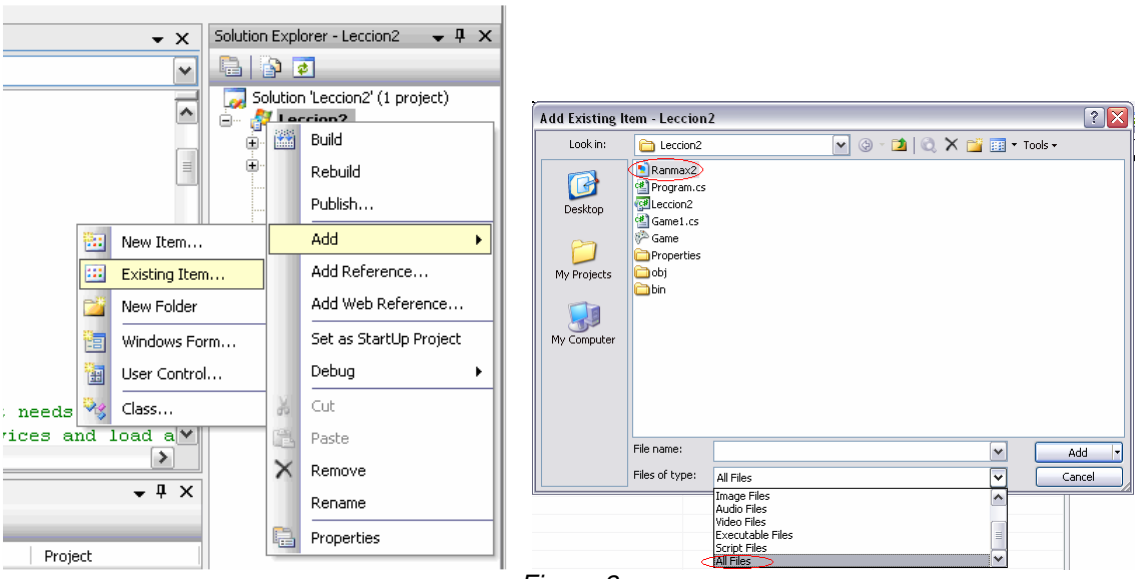


*Figura 3*
*Agregar un elemento al proyecto*
*Izquierda: pasos en el explorador de soluciones*
*Derecha: Ventana de añadir, seleccionar el tipo de archivo apropiado o "All Files"*

Por defecto el explorador de archivos solo dejara ver archivos de tipo ***.cs***, en la pestaña de abajo selecciona ver todos los archivos ***all files*** y busca tu imagen. Una vez le damos añadir ***add***, hemos agregado la imagen al proyecto exitosamente.

**Paso 3: Pintar la imagen:** A continuación vamos a agregar el código necesario para colocar la imagen en pantalla, para ello vamos a hacer uso de toda la potencia de XNA, de tal modo que sus librerías hagan el trabajo por nosotros.

1. Empecemos por agregar 2 variables en la clase (por defecto tiene el nombre de ***Game1***)

```
SpriteBatch batch;
Texture2D imagen;
```

*Copia estas 2 variables después del encabezado de la clase.*

2. Luego antes de continuar copia este método al final de la clase: Por ahora no nos interesa saber que hace este código en detalle, solo que carga la imagen por nosotros.

```
// Este código es para facilitar la carga de la imagen
protected void CargarImagen(string nombre)
{
    batch = new SpriteBatch(this.graphics.GraphicsDevice);
    ContentManager aLoader = new ContentManager(this.Services);
    imagen = aLoader.Load<Texture2D>(nombre) as Texture2D;
}
```

Continuando con la explicación anterior, las 2 variables (batch e imagen) que creamos antes, pertenecen a clases de las librerías de XNA, les hemos dado los nombres a nuestro gusto. La clase SpriteBtach se utiliza para dibujar imágenes, y la clase Texture2D se utiliza para cargar las imágenes que van a ser usadas. Cada juego que desee hacer uso de imágenes utilizará estas 2 clases, por lo que es importante aprender su uso. En esta lección solo exploraremos una pequeña parte de todo lo que se puede hacer con ellas.

3. A continuación podemos hacer uso de estas 2 variables, para lo cual vamos a agregar el siguiente código en el método "***LoadGraphicsContent***" que XNA creo previamente para nosotros.

```
protected override void LoadGraphicsContent(bool cargarTodo)
{
    if (cargarTodo)
    {
        CargarImagen("ranma");
    }
}
```

Del código anterior podemos rescatar las siguientes nuevas lecciones:

**Llamado a otros métodos:** note que en el interior de ***if*** hemos escrito ***CargarImagen***; si recuerdas ese es el nombre del método que antes colocamos, el cual carga la imagen por nosotros, lo único que necesita es el nombre de la imagen. Ten cuidado por que si escribes mal el nombre de la

imagen, el programa sacará un error de archivo no encontrado cuando lo ejecutes.

**El bloque de código "if(condición)":** La sentencia condicional "if(condición)" se usa para ejecutar un trozo se código de manera condicional, de modo que si la condición se cumple (***true***) entonces el código dentro de las llaves se ejecuta, de lo contrario, será omitido.

Ejemplos:

```
If(4 > 1){
    // Este código se ejecuta pues 4 es mayor que 1
}

If(2 != 2){
   // Este código no se ejecuta pues 2 no es diferente de 2
}

If(a < b){
   // Este código solo se ejecuta si el valor de
   // la variable "a" es menor que de la variable "b"
}

If(true){
   // Este código SIEMPRE se ejecutará sin
   // importar que pase, pues "true" significa verdadero
}

If(!true){
   // Este código NUNCA se ejecutará sin
   // importar que pase, pues "!" significa negar
   // por tanto negar a verdadero es igual a falso
}
```

4. Ahora que ya tenemos el código necesario para cargar la imagen, podemos pasar a dibujar la imagen en pantalla. Para poder hacer esto necesitamos agregar el código necesario en un método que hará este trabajo por nosotros. Por ahora no nos interesa saber que hace este código en detalle, solo que pinta la imagen por nosotros. En la clase ***Game1*** crea este método.

```
//Método creado para facilitar el código
protected void Pintar(Color fondo, int x, int y, int ancho, int alto,
  Color mascara, float angulo){
    graphics.GraphicsDevice.Clear(fondo);
    batch.Begin();
    batch.Draw(imagen, new Vector2(x, y), null, mascara, angulo,
     new Vector2(0, 0),1f, SpriteEffects.None, 0);
    batch.End();
}
```

5. En el método "***Draw***" que es otro de los métodos que XNA creo por nosotros inicialmente. Agrega el siguiente código al método "***Draw***".

```
protected override void Draw(GameTime gameTime)
{
   // pintar(color, x, y, ancho, alto, escala, ángulo)
   Pintar(Color.CornflowerBlue, 0, 0, 1f, Color.White, 0);
   base.Draw(gameTime);
}
```

Básicamente lo que se hace es llamar al método ***pintar*** para que el haga el trabajo difícil por nosotros, tu solo te debes preocupar por pasarle los parámetros correctos al método.

- **Color** = Color del fondo de la pantalla.
- **X** = Posición en el eje X, toma valores enteros.
- **Y** = Posición en el eje Y, toma valores enteros.
- **Escala** = Tamaño de la imagen; 1 = 100%, valores menores que 1 la encogen y valores mayores que uno la agrandan. Puede tomar valores enteros o flotantes[1].
- **Mascara** = Color que se le quiere mezclar a la imagen. Una mascara blanca deja la imagen con sus colores originales.
- **Ángulo** = Rotación en radianes que se le quiere dar a la imagen, puede tomar valores enteros o flotantes.

Del código anterior, podemos sacar las siguientes lecciones:

**El indicador de fin de instrucción:** Si observamos el código anterior, puede notarse que al final de cada línea se escribe el carácter punto y coma "**;**". Este carácter es usado para indicarle al compilador de C# que hemos terminado una instrucción; en otras palabras, toda instrucción debe terminar con el punto y coma, de lo contrario el compilador no sabrá cuando termina la instrucción e indicará un error en tiempo de compilación[2].

**Los comentarios:** Muchas veces queremos escribir el significado de una instrucción para luego recordarnos o explicar a otros que fue lo que se realizó. Los comentarios son la forma de hacer esto. De forma predeterminada, el compilador de C# ignorará toda línea que sea marcada como cometario por lo que no afectará a nuestro programa en tiempo de ejecución. Existen 2 tipos básicos de comentarios, los de línea sencilla (//):

```
// Comentario de línea sencilla (1 solo renglón)
```

Y los comentarios multilínea (/* */)

```
/* Este comentario puede ocupar
tantas líneas como se quiera
siempre que este entre los indicadores
de comentario multilínea.
*/
```

[1] Un flotantes es un valor con puntos decimal, los flotantes se indican con una ***f*** al final, Ej: float a = 5.5f;
[2] Existen básicamente 2 tiempos cuando se programa, el tiempo de compilación, cuando el código es interpretado, y tiempo de ejecución, cuando el código se encuentra en ejecución.

Es posible que te encuentres con otros tipos de comentacios (triple slash ///). Estos de momento no nos interezan, pues son de uso más especializado.

Si ejecutamos la aplicación (***F5***) obtendremos la siguiente pantalla (figura 4).

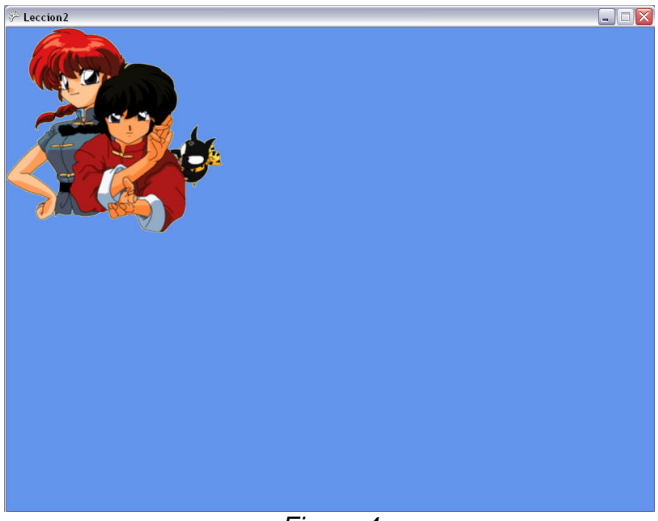


*Figura 4*
*Una imagen en pantalla*

Una pregunta natural que puede surgir es ¿porque la imagen apareció en la parte superior izquierda si se había indicado una posición (X,Y) de (0,0)?
Pues bien, esto se debe a que a diferencia del eje cartesiano al que estamos acostumbrados, c# no coloca el origen del plano, en el centro de la pantalla, sino en la esquina superior izquierda e invierte el eje de las Y, de tal modo que los valores positivos van hacia abajo (ver figura 5).

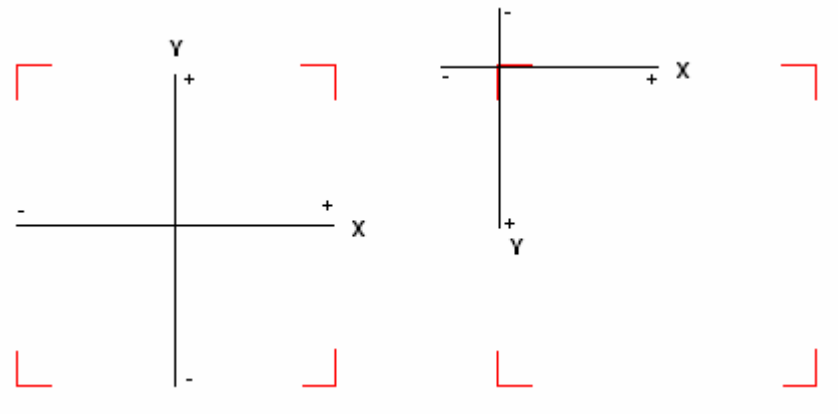


*Figura 5*
*A la izquierda un plano cartesiano tradicional, a la derecha un plano cartesiano de la forma como se interpreta en XNA*

Antes de continuar, practica un poco los siguientes ejercicios (figura 6):

- Trata de colocar la imagen en el centro de la pantalla.
- Cambia el tamaño de la imagen.
- Cambia el color de mascara y observa los resultados.

- Cambia el color del fondo de la ventana por otro diferente al azul.

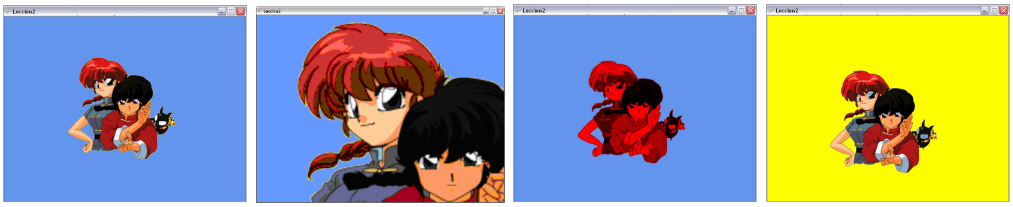

*Figura 6*
*Ejercicios*

Ahora que ya tenemos mas claro como pintar una imagen en la pantalla, ¿por que no le damos algo de movimiento?

**Paso 4: Animar la imagen:**

Lo que vamos a hacer ahora es trasladar la imagen en la pantalla a través de instrucciones de código sencillas.

1. Para comenzar vamos a necesitar una variable que nos guarde la posición de la imagen en todo momento (las coordenadas X,Y). por lo tanto ve a la clase ***Game1*** y agrega el siguiente atributo. Traslaciones.

```
//traslaciones
Vector2 posicion;
```

2. Ahora, lo que podemos hacer es hacer que en cada momento, las coordenadas X,Y de la imagen se vayan incrementando en una unidad. El método ***update*** que como mencionamos antes es uno de los que se llama automáticamente en todo momento es un buen lugar para colocar estos incrementos (nuevas líneas en negrita).

```
protected override void Update(GameTime gameTime){
    if(GamePad.GetState(PlayerIndex.One).Buttons.Back ==
     ButtonState.Pressed)
        this.Exit();

    posicion.X += 1f;
    posicion.Y += 1f;

    base.Update(gameTime);
}
```

3. finalmente tenemos que usar estas coordenadas cuando pintamos la imagen, para ello visita el método ***Draw*** y realiza estos cambios (cambios en negrita).

```
protected override void Draw(GameTime gameTime){
   // pintar(color, x, y, escala, mascara, ángulo)
   Pintar(Color.CornflowerBlue, posicion.X, posicion.Y, 1f,
     Color.White, 0);
   base.Draw(gameTime);
}
```

Si ejecutas el juego en este punto pudras ver como la imagen se desplaza por la pantalla lentamente hasta que la abandona.

4. Podríamos hacer que este proceso se repita indefinidamente agregando una condición que resetee el valor de las coordenadas X,Y cuando estas son muy grandes, visita el método ***update*** y realiza estos cambios (en negrita).

```
posicion.X += 1f;
posicion.Y += 1f;

if(posicion.X > 600f){
   posicion.X = 0f;
   posicion.Y = 0f;
}
```

Esto es solo una sugerencia, tú puedes inventar tu propio código para mover por la pantalla la imagen, por ejemplo en línea recta o que describa alguna ruta continuamente como un cuadrado por ejemplo.

5. otra cosa que podemos hacer es hacer rotar la imagen a medida que se desplaza. Para ello usaremos una variable que va a contener el ángulo de giro en todo momento. Para esto ve a la clase ***Game1*** y crea el siguiente atributo (cambios en negrita).

```
//traslaciones y rotciones
Vector2 posicion;
float angulo = 0f;
```

6. Incrementemos de a poco el ángulo en cada momento y observa como se comporta la imagen al ejecutar. Primero ve al método ***update*** y coloca allí el incremento (cambios en negrita).

```
posicion.X += 1f;
posicion.Y += 1f;
angulo += 0.04f;

if(posicion.X > 600f){
   posicion.X = 0f;
   posicion.Y = 0f;
}
```

7. finalmente usa la variable del ángulo cuando pintamos la imagen. Visita el método ***Draw*** (cambio en negrita).

```
// pintar(color, x, y, escala, mascara, ángulo)
Pintar(Color.CornflowerBlue, posicion.X, posicion.Y, 1f,
  Color.White, angulo);
```

Puedes notar que el punto de rotación se ha fijado por defecto no en el centro de la imagen sino en su esquina superior izquierda. Realizar los cambios necesarios para ajustar el centro de rotación en el centro es un ejercicio que dejo para los curiosos.