Programming Two

Tutorial 7: C# and XNA: Input and

Introduction

This tutorial builds upon last week, getting us to add keyboard input to our code.

Stage One: Adding Keyboard Input

If you did not add keyboard input at the end of last week now is the time.
Create the following function:

```
protected void UpdateInput()
{
    KeyboardState newState = Keyboard.GetState();

    // Is the SPACE key down?
    if (newState.IsKeyDown(Keys.Space))
    {
        // If not down last update, key has just been pressed.
        if (!oldState.IsKeyDown(Keys.Space))
        {
           // do stuff here
        }
    }
    else if (oldState.IsKeyDown(Keys.Space))
    {
        // Key was down last update, but not down now, so
        // it has just been released.
    }
    else if (newState.IsKeyDown(Keys.Escape))
    {
        this.Exit();
    }

    // Update saved state.
    oldState = newState;
}
```

You need to add `KeyboardState oldState` as a class member data.

Stage Two: Animation Through Your Keyboard

Call your UpdateSprite() function from your keyboard class to animate, change frames, of your sprite when you press space.

Stage Three: Movement of our Sprite

Change your update functions to enable you to walk your sprite around the window.
Prevent the sprite from walking off the edge of the screen.
You need to perform a simple check to prevent incrementing the sprites position > than the bounds of the screen.
Look at the msdn site for graphicsdevice and rectangle. They provide some handing member functions. For instance you can access the width and height of the window via the following function:

```
device.Viewport.Width;
```

Stage Four: Adding a Background

This is a simple as drawing a sprite with its dimensions set to the size of the display area.

Stage Five: Transparency

You need to specify transparency in the image and then call alpha blend:

```
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
```

When drawing the sprite.

As for file types:
Png's are better because of the compression. It doesn't matter which colour is transparent as long as one is. You can create an image in Photoshop, make sure you can see the chequered background, do a "Save For Web" and choose PNG-24 as the output type. You can even create gradient transparencies with 24 bit PNG's to give different levels of transparency.

As long as you use spriteBatch.Begin(SpriteBlendMode.AlphaBlend); it will pick up the transparency from the .PNG. I was using the DX Texture tool but there is no need, it's just an extra step of messing around that isn't necessary, certainly not for the 2D stuff anyway.

If you want smaller file sizes, choose the PNG-8 option in the "Save For Web" step and then you can then choose a lower number of colours.

BMP's are ridiculously large on file size, JPG's don't support transparency and Gifs often have issues with the normal import routes associated with XNA.
The other file type to use, other than PNG, is DDS. Using this method you can use DirectX Texture Tool to create a DDS file. Add a mask to the alpha channel, and your regular image to the surface.

Stage Six: Coding a Sprite Class

You should now try to code a sprite class.
First you need to think of the key properties of the sprite class and how it will interact with the game.