XNA 4.0 RPG Tutorials

Part 25

Level Editor Continued

I'm writing these tutorials for the new XNA 4.0 framework. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the XNA 4.0 RPG tutorials page of my web site. I will be making my version of the project available for download at the end of each tutorial. It will be included on the page that links to the tutorials.

I'm going to work on the level editor again in this tutorial and add in some more features. I found a bug while I was working on the editor though. If you open a map and try and add a new layer to it a Null Reference exception will be thrown. I forget to set the **levelData** field. All I do is set the field **levelData** to the **newLevel** that I read in. Change the event handler for opening a level to the following.

```
void openLevelToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog ofDialog = new OpenFileDialog();
    ofDialog.Filter = "Level Files (*.xml)|*.xml";
    ofDialog.CheckFileExists = true;
    ofDialog.CheckPathExists = true;

    DialogResult result = ofDialog.ShowDialog();

    if (result != DialogResult.OK)
        return;

    string path = Path.GetDirectoryName(ofDialog.FileName);

    LevelData newLevel = null;
    MapData mapData = null;

    try
    {
        newLevel = XnaSerializer.Deserialize<LevelData>(ofDialog.FileName);
        mapData = XnaSerializer.Deserialize<MapData>(path + @"\Maps\" + newLevel.MapName +
".xml");
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Error reading level");
        return;
    }

    tileSetImages.Clear();
    tileSetData.Clear();
    tileSets.Clear();
    layers.Clear();
    lbTileset.Items.Clear();
    clbLayers.Items.Clear();

    levelData = newLevel;

    foreach (TilesetData data in mapData.Tilesets)
    {
        Texture2D texture = null;

        tileSetData.Add(data);
```

```
        lbTileset.Items.Add(data.TilesetName);

        GDIImage image = (GDIImage)GDIBitmap.FromFile(data.TilesetImageName);
        tileSetImages.Add(image);

        using (Stream stream = new FileStream(data.TilesetImageName, FileMode.Open,
FileAccess.Read))
        {
            texture = Texture2D.FromStream(GraphicsDevice, stream);
            tileSets.Add(
                new Tileset(
                    texture,
                    data.TilesWide,
                    data.TilesHigh,
                    data.TileWidthInPixels,
                    data.TileHeightInPixels));
        }
    }

    foreach (MapLayerData data in mapData.Layers)
    {
        clbLayers.Items.Add(data.MapLayerName, true);
        layers.Add(MapLayer.FromMapLayerData(data));
    }

    lbTileset.SelectedIndex = 0;
    clbLayers.SelectedIndex = 0;
    nudCurrentTile.Value = 0;

    map = new TileMap(tileSets, layers);

    tilesetToolStripMenuItem.Enabled = true;
    mapLayerToolStripMenuItem.Enabled = true;
    charactersToolStripMenuItem.Enabled = true;
    chestsToolStripMenuItem.Enabled = true;
    keysToolStripMenuItem.Enabled = true;
}
```

I saw on my forum that the editor was being a little sluggish. I'm going to fix that first. You can easily change the **Interval** property of **controlTimer** so that the **Tick** event is triggered more often so that the logic and drawing will be done more often. The problem with doing that is that the map will scroll really fast. What I did to fix that was introduce a frame counter and only call the scrolling logic every so often.

To the **Field** region you are going to want to add in a field to keep track of the frame count. Add in the following field.

```
int frameCount = 0;
```

The next thing you will want to do is to change the **Interval** property of **controlTimer** so that the **Tick** event will be called more often. If you set it to 17 milliseconds it will be called approximately 60 times per second. I set the value in the **FormMain_Load** method so change that method to the following.

```
void FormMain_Load(object sender, EventArgs e)
{
    lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
    nudCurrentTile.ValueChanged += new EventHandler(nudCurrentTile_ValueChanged);

    Rectangle viewPort = new Rectangle(0, 0, mapDisplay.Width, mapDisplay.Height);
    camera = new Camera(viewPort);

    engine = new Engine(32, 32);
```

```
    controlTimer.Tick += new EventHandler(controlTimer_Tick);
    controlTimer.Enabled = true;
    controlTimer.Interval = 17;

    tbMapLocation.TextAlign = HorizontalAlignment.Center;
    pbTilesetPreview.MouseDown += new MouseEventHandler(pbTilesetPreview_MouseDown);

    mapDisplay.SizeChanged += new EventHandler(mapDisplay_SizeChanged);
}
```

The next step will be to update the **frameCount** field in the event handler for the **Tick** event. Change the handler to the following.

```
void controlTimer_Tick(object sender, EventArgs e)
{
    frameCount = ++frameCount % 6;
    mapDisplay.Invalidate();
    Logic();
}
```

What I do is pre-increment **frameCount** and get the remainder of dividing it by 6. So what happens is that **frameCount** will have the values 0, 1, 2, 3, 4, and 5. I will check in the **Logic** method when the value is 0 that the map should be scrolled. Change the **Logic** method to the following.

```
private void Logic()
{
    if (layers.Count == 0)
        return;

    Vector2 position = camera.Position;

    if (trackMouse)
    {
        if (frameCount == 0)
        {
            if (mouse.X < Engine.TileWidth)
                position.X -= Engine.TileWidth;

            if (mouse.X > mapDisplay.Width - Engine.TileWidth)
                position.X += Engine.TileWidth;

            if (mouse.Y < Engine.TileHeight)
                position.Y -= Engine.TileHeight;

            if (mouse.Y > mapDisplay.Height - Engine.TileHeight)
                position.Y += Engine.TileHeight;

            camera.Position = position;
            camera.LockCamera();
        }

        position.X = mouse.X + camera.Position.X;
        position.Y = mouse.Y + camera.Position.Y;

        Point tile = Engine.VectorToCell(position);

        tbMapLocation.Text =
            "( " + tile.X.ToString() + ", " + tile.Y.ToString() + " )";

        if (isMouseDown)
        {
            if (rbDraw.Checked)
            {
                layers[clbLayers.SelectedIndex].SetTile(
                    tile.X,
                    tile.Y,
```

```
                    (int)nudCurrentTile.Value,
                    lbTileset.SelectedIndex);
            }
            if (rbErase.Checked)
            {
                layers[clbLayers.SelectedIndex].SetTile(
                    tile.X,
                    tile.Y,
                    -1,
                    -1);
            }
        }
    }

}
```

All I did was wrap the logic for scrolling the map in an if statement. If the **frameCount** field is at 0 then I want to check to see if the map should be scrolled. This will fix the problem with the editor lagging and it still scrolling at a reasonable rate.

What I want to do now is to add in different brush sizes. What I did was add in a new menu item, **&Brushes**, and under **&Brushes** I added: **1 x 1, 2 x 2**, **4 x 4**, and **8 x 8**. I also set the **Checked** property of **1 x 1** to **True**. You are going to want to subscribe to the **Click** event of the menu items. I did that in the constructor. I also placed the handlers into a region of their own, add in the region below as well. Also add in the following field to the **Field** region.

```
int brushWidth = 1;

public FormMain()
{
    InitializeComponent();

    this.Load += new EventHandler(FormMain_Load);
    this.FormClosing += new FormClosingEventHandler(FormMain_FormClosing);

    tilesetToolStripMenuItem.Enabled = false;
    mapLayerToolStripMenuItem.Enabled = false;
    charactersToolStripMenuItem.Enabled = false;
    chestsToolStripMenuItem.Enabled = false;
    keysToolStripMenuItem.Enabled = false;

    newLevelToolStripMenuItem.Click += new EventHandler(newLevelToolStripMenuItem_Click);
    newTilesetToolStripMenuItem.Click += new EventHandler(newTilesetToolStripMenuItem_Click);
    newLayerToolStripMenuItem.Click += new EventHandler(newLayerToolStripMenuItem_Click);

    saveLevelToolStripMenuItem.Click += new EventHandler(saveLevelToolStripMenuItem_Click);

    openLevelToolStripMenuItem.Click += new EventHandler(openLevelToolStripMenuItem_Click);

    mapDisplay.OnInitialize += new EventHandler(mapDisplay_OnInitialize);
    mapDisplay.OnDraw += new EventHandler(mapDisplay_OnDraw);

    x1ToolStripMenuItem.Click += new EventHandler(x1ToolStripMenuItem_Click);
    x2ToolStripMenuItem.Click += new EventHandler(x2ToolStripMenuItem_Click);
    x4ToolStripMenuItem.Click += new EventHandler(x4ToolStripMenuItem_Click);
    x8ToolStripMenuItem.Click += new EventHandler(x8ToolStripMenuItem_Click);
}

#region Brush Event Handler Region

void x1ToolStripMenuItem_Click(object sender, EventArgs e)
{
    x1ToolStripMenuItem.Checked = true;
    x2ToolStripMenuItem.Checked = false;
    x4ToolStripMenuItem.Checked = false;
```

```
    x8ToolStripMenuItem.Checked = false;

    brushWidth = 1;
}

void x2ToolStripMenuItem_Click(object sender, EventArgs e)
{
    x1ToolStripMenuItem.Checked = false;
    x2ToolStripMenuItem.Checked = true;
    x4ToolStripMenuItem.Checked = false;
    x8ToolStripMenuItem.Checked = false;

    brushWidth = 2;
}

void x4ToolStripMenuItem_Click(object sender, EventArgs e)
{
    x1ToolStripMenuItem.Checked = false;
    x2ToolStripMenuItem.Checked = false;
    x4ToolStripMenuItem.Checked = true;
    x8ToolStripMenuItem.Checked = false;

    brushWidth = 4;
}

void x8ToolStripMenuItem_Click(object sender, EventArgs e)
{
    x1ToolStripMenuItem.Checked = false;
    x2ToolStripMenuItem.Checked = false;
    x4ToolStripMenuItem.Checked = false;
    x8ToolStripMenuItem.Checked = true;

    brushWidth = 8;
}

#endregion
```

The handlers all have the same basic form. They set the **Checked** property of the other menu items to false and theirs to true. They also set the **brushWidth** field to be the appropriate width.

Time to update the logic a little and try drawing of the display. In the logic method I called a new method that I wrote called **SetTiles** that will set a range of tiles based on the **brushWidth** field. Change the **Logic** method to the following and add the **SetTiles** method below it.

```
private void Logic()
{
    if (layers.Count == 0)
        return;

    Vector2 position = camera.Position;

    if (trackMouse)
    {
        if (frameCount == 0)
        {
            if (mouse.X < Engine.TileWidth)
                position.X -= Engine.TileWidth;

            if (mouse.X > mapDisplay.Width - Engine.TileWidth)
                position.X += Engine.TileWidth;

            if (mouse.Y < Engine.TileHeight)
                position.Y -= Engine.TileHeight;

            if (mouse.Y > mapDisplay.Height - Engine.TileHeight)
                position.Y += Engine.TileHeight;
```

```
            camera.Position = position;
            camera.LockCamera();
        }

        position.X = mouse.X + camera.Position.X;
        position.Y = mouse.Y + camera.Position.Y;

        Point tile = Engine.VectorToCell(position);

        tbMapLocation.Text =
            "( " + tile.X.ToString() + ", " + tile.Y.ToString() + " )";

        if (isMouseDown)
        {
            if (rbDraw.Checked)
                SetTiles(tile, (int)nudCurrentTile.Value, lbTileset.SelectedIndex);

            if (rbErase.Checked)
                SetTiles(tile, -1, -1);
        }
    }
}

private void SetTiles(Point tile, int tileIndex, int tileset)
{
    int selected = clbLayers.SelectedIndex;

    for (int y = 0; y < brushWidth; y++)
    {
        if (tile.Y + y >= layers[selected].Height)
            break;

        for (int x = 0; x < brushWidth; x++)
        {
            if (tile.X + x < layers[selected].Width)
                layers[selected].SetTile(
                    tile.X + x,
                    tile.Y + y,
                    tileIndex,
                    tileset);
        }
    }
}
```

The **Logic** method will call the **SetTiles** method passing in the **Point tile**, the **Value** property of **nudCurrentTile**, and the **SelectedIndex** of **lbTileset** if **rbDraw** is checked. If **rbErase** is checked I pass in -1 for the tile and -1 for the tileset.

The **SetTiles** method first declared a local variable **selected** that is the **SelectedIndex** of **clbLayers**, the current layer to be drawn to. I then have a set of nested for loops. The outer loop, **y**, loops from 0 to **brushWidth**. There is a check to see if **tile.Y + y** is greater than or equal to the **Height** of the layer. If it is the index is outside the height of the map and you don't want to continue drawing so I break out of the loop. In the inner loop, **x**, I loop from 0 to **brushWidth**. If **tile.X + x** is less than the **Width** of the layer I call the **SetTile** method of the layer passing in **tile.X + x**, **tile.Y + y**, **tileIndex** and **tileset**.

I want to update the **DrawDisplay** method. First though I want to add in a few things. In the **View** menu I want to add in being able to draw the grid in different colors. I created the grid image to be all white. This means if I use color different than **Color.White** as the tint color the grid will be that color. I'm going to add a menu with a sub menu to the side with different color options. Under **&Display Grid** add in **&Grid Color**. Now, select **Grid Color** and beside it add in the following items. **&Black**, **B&lue**, **R&ed**, **&Green**, **&Yellow**, **&White**. For the **White** option I set **Checked** to true. You can add

in other colors if you want and set your default to what you want.

Now, in the constructor I want to wire some event handlers and add in a new region for the handlers. I also want to add in a field. Add in this field, change the constructor to the following, and add in this region.

```
Color gridColor = Color.White;

public FormMain()
{
    InitializeComponent();

    this.Load += new EventHandler(FormMain_Load);
    this.FormClosing += new FormClosingEventHandler(FormMain_FormClosing);

    tilesetToolStripMenuItem.Enabled = false;
    mapLayerToolStripMenuItem.Enabled = false;
    charactersToolStripMenuItem.Enabled = false;
    chestsToolStripMenuItem.Enabled = false;
    keysToolStripMenuItem.Enabled = false;

    newLevelToolStripMenuItem.Click += new EventHandler(newLevelToolStripMenuItem_Click);
    newTilesetToolStripMenuItem.Click += new EventHandler(newTilesetToolStripMenuItem_Click);
    newLayerToolStripMenuItem.Click += new EventHandler(newLayerToolStripMenuItem_Click);

    saveLevelToolStripMenuItem.Click += new EventHandler(saveLevelToolStripMenuItem_Click);

    openLevelToolStripMenuItem.Click += new EventHandler(openLevelToolStripMenuItem_Click);

    mapDisplay.OnInitialize += new EventHandler(mapDisplay_OnInitialize);
    mapDisplay.OnDraw += new EventHandler(mapDisplay_OnDraw);

    x1ToolStripMenuItem.Click += new EventHandler(x1ToolStripMenuItem_Click);
    x2ToolStripMenuItem.Click += new EventHandler(x2ToolStripMenuItem_Click);
    x4ToolStripMenuItem.Click += new EventHandler(x4ToolStripMenuItem_Click);
    x8ToolStripMenuItem.Click += new EventHandler(x8ToolStripMenuItem_Click);

    blackToolStripMenuItem.Click += new EventHandler(blackToolStripMenuItem_Click);
    blueToolStripMenuItem.Click += new EventHandler(blueToolStripMenuItem_Click);
    redToolStripMenuItem.Click += new EventHandler(redToolStripMenuItem_Click);
    greenToolStripMenuItem.Click += new EventHandler(greenToolStripMenuItem_Click);
    yellowToolStripMenuItem.Click += new EventHandler(yellowToolStripMenuItem_Click);
    whiteToolStripMenuItem.Click += new EventHandler(whiteToolStripMenuItem_Click);
}

#region Grid Color Event Handler Region

void whiteToolStripMenuItem_Click(object sender, EventArgs e)
{
    gridColor = Color.White;
    blackToolStripMenuItem.Checked = false;
    blueToolStripMenuItem.Checked = false;
    redToolStripMenuItem.Checked = false;
    greenToolStripMenuItem.Checked = false;
    yellowToolStripMenuItem.Checked = false;
    whiteToolStripMenuItem.Checked = true;
}

void yellowToolStripMenuItem_Click(object sender, EventArgs e)
{
    gridColor = Color.Yellow;
    blackToolStripMenuItem.Checked = false;
    blueToolStripMenuItem.Checked = false;
    redToolStripMenuItem.Checked = false;
    greenToolStripMenuItem.Checked = false;
    yellowToolStripMenuItem.Checked = true;
    whiteToolStripMenuItem.Checked = false;
```

```
}

void greenToolStripMenuItem_Click(object sender, EventArgs e)
{
    gridColor = Color.Green;
    blackToolStripMenuItem.Checked = false;
    blueToolStripMenuItem.Checked = false;
    redToolStripMenuItem.Checked = false;
    greenToolStripMenuItem.Checked = true;
    yellowToolStripMenuItem.Checked = false;
    whiteToolStripMenuItem.Checked = false;
}

void redToolStripMenuItem_Click(object sender, EventArgs e)
{
    gridColor = Color.Red;
    blackToolStripMenuItem.Checked = false;
    blueToolStripMenuItem.Checked = false;
    redToolStripMenuItem.Checked = true;
    greenToolStripMenuItem.Checked = false;
    yellowToolStripMenuItem.Checked = false;
    whiteToolStripMenuItem.Checked = false;
}

void blueToolStripMenuItem_Click(object sender, EventArgs e)
{
    gridColor = Color.Blue;
    blackToolStripMenuItem.Checked = false;
    blueToolStripMenuItem.Checked = true;
    redToolStripMenuItem.Checked = false;
    greenToolStripMenuItem.Checked = false;
    yellowToolStripMenuItem.Checked = false;
    whiteToolStripMenuItem.Checked = false;
}

void blackToolStripMenuItem_Click(object sender, EventArgs e)
{
    gridColor = Color.Black;
    blackToolStripMenuItem.Checked = true;
    blueToolStripMenuItem.Checked = false;
    redToolStripMenuItem.Checked = false;
    greenToolStripMenuItem.Checked = false;
    yellowToolStripMenuItem.Checked = false;
    whiteToolStripMenuItem.Checked = false;
}

#endregion
```

The handlers should look familiar as they have the same form as those for the brush sizes. I set the **gridColor** field to the appropriate color. I then set the **Checked** property for the appropriate item to true and the others to false. You can now update the **DrawDisplay** method to the following.

```
private void DrawDisplay()
{
    if (map == null)
        return;

    Rectangle destination = new Rectangle(
        0,
        0,
        Engine.TileWidth,
        Engine.TileHeight);

    if (displayGridToolStripMenuItem.Checked)
    {
        int maxX = mapDisplay.Width / Engine.TileWidth + 1;
        int maxY = mapDisplay.Height / Engine.TileHeight + 1;
```

```
        spriteBatch.Begin();

        for (int y = 0; y < maxY; y++)
        {
            destination.Y = y * Engine.TileHeight;

            for (int x = 0; x < maxX; x++)
            {
                destination.X = x * Engine.TileWidth;

                spriteBatch.Draw(grid, destination, gridColor);
            }
        }

        spriteBatch.End();
    }

    spriteBatch.Begin();

    destination.X = mouse.X;
    destination.Y = mouse.Y;

    if (rbDraw.Checked)
    {
        spriteBatch.Draw(
            tileSets[lbTileset.SelectedIndex].Texture,
            destination,
            tileSets[lbTileset.SelectedIndex].SourceRectangles[(int)nudCurrentTile.Value],
            Color.White);
    }

    spriteBatch.Draw(cursor, destination, Color.White);

    spriteBatch.End();
}
```

The change is that instead of using **Color.White** in the call to **Draw** when I'm drawing the grid. I also added in an if statement to check to see if the **Checked** property of **rbDraw** is true. If it is true I draw the preview tile.

There is one more thing that you might want to add to the editor. You may want to have the tiles that will be changed high lighted. The first step in doing that is to that is to have a **Texture2D**. In this case I'm not going to create an image in an image editor and add it to the content. I'm going to create a **Texture2D** with code. Add in these fields to the **Field** region and change **mapDisplay_OnInitialize** to the following.

```
Texture2D shadow;
Vector2 shadowPosition = Vector2.Zero;

void mapDisplay_OnInitialize(object sender, EventArgs e)
{
    spriteBatch = new SpriteBatch(GraphicsDevice);

    shadow = new Texture2D(GraphicsDevice, 20, 20, false, SurfaceFormat.Color);

    Color[] data = new Color[shadow.Width * shadow.Height];
    Color tint = Color.LightSteelBlue;
    tint.A = 25;

    for (int i = 0; i < shadow.Width * shadow.Height; i++)
        data[i] = tint;

    shadow.SetData<Color>(data);

    mapDisplay.MouseEnter += new EventHandler(mapDisplay_MouseEnter);
```

```
    mapDisplay.MouseLeave += new EventHandler(mapDisplay_MouseLeave);
    mapDisplay.MouseMove += new MouseEventHandler(mapDisplay_MouseMove);
    mapDisplay.MouseDown += new MouseEventHandler(mapDisplay_MouseDown);
    mapDisplay.MouseUp += new MouseEventHandler(mapDisplay_MouseUp);

    try
    {
        using (Stream stream = new FileStream(@"Content\grid.png", FileMode.Open,
FileAccess.Read))
        {
            grid = Texture2D.FromStream(GraphicsDevice, stream);
            stream.Close();
        }

        using (Stream stream = new FileStream(@"Content\cursor.png", FileMode.Open,
FileAccess.Read))
        {
            cursor = Texture2D.FromStream(GraphicsDevice, stream);
            stream.Close();
        }
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Error reading images");
        grid = null;
        cursor = null;
    }
}
```

What I did is add a **Texture2D** field and a **Vector2** field. The **Texture2D** is the image for the shadow and the **Vector2** will be its position on the map, not the display, the map itself. In the event handler for the **OnInitialize** method of **mapDisplay** I create a new **Texture2D** passing in the **GraphicsDevice** for the map display, 20 for the width and height, and **SurfaceFormat.Color**. That format is a 32-bit format with 8 bits for red, blue, green, and alpha. I then create an array the size of the width **shadow** times the height of **shadow**. I create a color, **tint**, set to **Blue** with a low alpha channel. In a for loop I loop over the entire array and set each item to be **tint**. I then call the **SetData** method that takes a **Color** and pass in the array of color.

You will want to update the **Logic** method to update the **shadowPosition** field so that it holds the tile that the mouse is hovering over. All I did was after finding the tile the mouse plus the camera is in is set **shadowPosition** to a **Vector2** with **X** and **Y** components **tile.X** and **tile.Y**. Update the **Logic** method to the following.

```
private void Logic()
{
    if (layers.Count == 0)
        return;

    Vector2 position = camera.Position;

    if (trackMouse)
    {
        if (frameCount == 0)
        {
            if (mouse.X < Engine.TileWidth)
                position.X -= Engine.TileWidth;

            if (mouse.X > mapDisplay.Width - Engine.TileWidth)
                position.X += Engine.TileWidth;

            if (mouse.Y < Engine.TileHeight)
                position.Y -= Engine.TileHeight;
```

```
            if (mouse.Y > mapDisplay.Height - Engine.TileHeight)
                position.Y += Engine.TileHeight;

            camera.Position = position;
            camera.LockCamera();
        }

        position.X = mouse.X + camera.Position.X;
        position.Y = mouse.Y + camera.Position.Y;

        Point tile = Engine.VectorToCell(position);
        shadowPosition = new Vector2(tile.X, tile.Y);

        tbMapLocation.Text =
            "( " + tile.X.ToString() + ", " + tile.Y.ToString() + " )";

        if (isMouseDown)
        {
            if (rbDraw.Checked)
                SetTiles(tile, (int)nudCurrentTile.Value, lbTileset.SelectedIndex);

            if (rbErase.Checked)
                SetTiles(tile, -1, -1);
        }
    }
}
```

Now I will update the **Render** method. What it will do is draw the **shadow** texture tinted white with a low alpha value so it is somewhat transparent.

```
private void Render()
{
    for (int i = 0; i < layers.Count; i++)
    {
        spriteBatch.Begin(
            SpriteSortMode.Deferred,
            BlendState.AlphaBlend,
            SamplerState.PointClamp,
            null,
            null,
            null,
            camera.Transformation);

        if (clbLayers.GetItemChecked(i))
            layers[i].Draw(spriteBatch, camera, tileSets);

        Rectangle destination = new Rectangle(
            (int)shadowPosition.X * Engine.TileWidth,
            (int)shadowPosition.Y * Engine.TileHeight,
            brushWidth * Engine.TileWidth,
            brushWidth * Engine.TileHeight);

        Color tint = Color.White;
        tint.A = 1;

        spriteBatch.Draw(shadow, destination, tint);
        spriteBatch.End();
    }

    DrawDisplay();
}
```

The one thing I want to do is to add in writing out and reading in tilesets and map layers. This way you don't have to always be creating tilesets when you are reusing the same tileset over and over again. You can also have a stock layer filled with grass that you can add easily. As always the first step is to wire event handlers. I went a head and wired handlers for the **Click** event of both the **Tileset** menu and the

Map Layer menu. I did that in the constructor for **FormMain**, of course. I added the handlers for saving to the **Save Menu Event Handler** region and for opening to the **Open Menu Event Handler** region. Change the constructor of **FormMain** to the following first.

```
public FormMain()
{
    InitializeComponent();

    this.Load += new EventHandler(FormMain_Load);
    this.FormClosing += new FormClosingEventHandler(FormMain_FormClosing);

    tilesetToolStripMenuItem.Enabled = false;
    mapLayerToolStripMenuItem.Enabled = false;
    charactersToolStripMenuItem.Enabled = false;
    chestsToolStripMenuItem.Enabled = false;
    keysToolStripMenuItem.Enabled = false;

    newLevelToolStripMenuItem.Click += new EventHandler(newLevelToolStripMenuItem_Click);
    newTilesetToolStripMenuItem.Click += new EventHandler(newTilesetToolStripMenuItem_Click);
    newLayerToolStripMenuItem.Click += new EventHandler(newLayerToolStripMenuItem_Click);

    saveLevelToolStripMenuItem.Click += new EventHandler(saveLevelToolStripMenuItem_Click);

    openLevelToolStripMenuItem.Click += new EventHandler(openLevelToolStripMenuItem_Click);

    mapDisplay.OnInitialize += new EventHandler(mapDisplay_OnInitialize);
    mapDisplay.OnDraw += new EventHandler(mapDisplay_OnDraw);

    x1ToolStripMenuItem.Click += new EventHandler(x1ToolStripMenuItem_Click);
    x2ToolStripMenuItem.Click += new EventHandler(x2ToolStripMenuItem_Click);
    x4ToolStripMenuItem.Click += new EventHandler(x4ToolStripMenuItem_Click);
    x8ToolStripMenuItem.Click += new EventHandler(x8ToolStripMenuItem_Click);

    blackToolStripMenuItem.Click += new EventHandler(blackToolStripMenuItem_Click);
    blueToolStripMenuItem.Click += new EventHandler(blueToolStripMenuItem_Click);
    redToolStripMenuItem.Click += new EventHandler(redToolStripMenuItem_Click);
    greenToolStripMenuItem.Click += new EventHandler(greenToolStripMenuItem_Click);
    yellowToolStripMenuItem.Click += new EventHandler(yellowToolStripMenuItem_Click);
    whiteToolStripMenuItem.Click += new EventHandler(whiteToolStripMenuItem_Click);

    saveTilesetToolStripMenuItem.Click += new EventHandler(saveTilesetToolStripMenuItem_Click);
    saveLayerToolStripMenuItem.Click += new EventHandler(saveLayerToolStripMenuItem_Click);

    openTilesetToolStripMenuItem.Click += new EventHandler(openTilesetToolStripMenuItem_Click);
    openLayerToolStripMenuItem.Click += new EventHandler(openLayerToolStripMenuItem_Click);
}
```

That just wires the event handlers. I will start with the handlers for the save events. Add these two methods to the **Save Menu Event Handlers** region.

```
void saveTilesetToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (tileSetData.Count == 0)
        return;

    SaveFileDialog sfDialog = new SaveFileDialog();
    sfDialog.Filter = "Tileset Data (*.tdat)|*.tdat";
    sfDialog.CheckPathExists = true;
    sfDialog.OverwritePrompt = true;
    sfDialog.ValidateNames = true;

    DialogResult result = sfDialog.ShowDialog();

    if (result != DialogResult.OK)
        return;
```

```
    try
    {
        XnaSerializer.Serialize<TilesetData>(
            sfDialog.FileName,
            tileSetData[lbTileset.SelectedIndex]);
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Error saving tileset");
    }
}

void saveLayerToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (layers.Count == 0)
        return;

    SaveFileDialog sfDialog = new SaveFileDialog();
    sfDialog.Filter = "Map Layer Data (*.mldt)|*.mldt";
    sfDialog.CheckPathExists = true;
    sfDialog.OverwritePrompt = true;
    sfDialog.ValidateNames = true;

    DialogResult result = sfDialog.ShowDialog();

    if (result != DialogResult.OK)
        return;

    MapLayerData data = new MapLayerData(
        clbLayers.SelectedItem.ToString(),
        layers[clbLayers.SelectedIndex].Width,
        layers[clbLayers.SelectedIndex].Height);

    for (int y = 0; y < layers[clbLayers.SelectedIndex].Height; y++)
    {
        for (int x = 0; x < layers[clbLayers.SelectedIndex].Width; x++)
        {
            data.SetTile(
                x,
                y,
                layers[clbLayers.SelectedIndex].GetTile(x, y).TileIndex,
                layers[clbLayers.SelectedIndex].GetTile(x, y).Tileset);
        }
    }

    try
    {
        XnaSerializer.Serialize<MapLayerData>(sfDialog.FileName, data);
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Error saving map layer data");
    }
}
```

The methods are very similar. The big difference is the data that they work on. You also need to do a little more work to save the layer data because I don't have a collection of **MapLayerData** to work with. First I check to make sure there is data to write out checking the **Count** property of the appropriate collection. I then create a **SaveFileDialog** object to display to the user. I set a few properties for the **SaveFileDialog** next. The **Filter** property filters the files shown and is different for **TilesetData** and **MapLayerData**. I also set a few fields to validate that a correct path is chosen and if the file exists there will be an overwrite prompt. I capture the result of the **ShowDialog** method of the **SaveFileDialog**. If the result was not that the **OK** button was pressed I exit out of the method. I then try to save the data.

In the handler for saving a tileset I already have the collection of **TilesetData** objects so I don't have to create one. In a try-catch block I call the **Serialize** method of **XnaSerializer** passing in the **FileName** property of the **SaveFileDialog** object and the **TilesetData** object in **tileSetData** that is currently selected in **lbTileset**. If there was an exception I display the exception in a message box with a title saying there was an error saving the tileset.

In the handler for saving a map layer I first have to create a **MapLayerData** object to work with. I create the object passing in the **SelectedItem** of **clbLayers** for the name of the layer and for the width of the map I use the **Width** and **Height** properties of the layer at the **SelectedIndex** property of **clbLayers**. There are nested for loops to loop through the entire layer. I call the **SetTile** method of **MapLayerData** passing in **x, y**, **GetTile(x, y).TileIndex**, and **GetTile(x, y).Tileset**. Then there is a try catch block where I try to serialize the object. If there is an exception I display the exception in a message box with a title saying there was an error saving the layer.

Reading in tilesets is a little more complicated than writing out as you recall from loading a level. So I will first cover reading in tilesets and then reading in map layers. Add the following method to the **Open Menu Event Handler** region.

```
void openTilesetToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog ofDialog = new OpenFileDialog();
    ofDialog.Filter = "Tileset Data (*.tdat)|*.tdat";
    ofDialog.CheckPathExists = true;
    ofDialog.CheckFileExists = true;

    DialogResult result = ofDialog.ShowDialog();

    if (result != DialogResult.OK)
        return;

    TilesetData data = null;
    Texture2D texture = null;
    Tileset tileset = null;
    GDIImage image = null;

    try
    {
        data = XnaSerializer.Deserialize<TilesetData>(ofDialog.FileName);
        using (Stream stream = new FileStream(data.TilesetImageName, FileMode.Open,
FileAccess.Read))
        {
            texture = Texture2D.FromStream(GraphicsDevice, stream);
            stream.Close();
        }

        image = (GDIImage)GDIBitmap.FromFile(data.TilesetImageName);

        tileset = new Tileset(
            texture,
            data.TilesWide,
            data.TilesHigh,
            data.TileWidthInPixels,
            data.TileHeightInPixels);
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Error reading tileset");
        return;
    }

    for (int i = 0; i < lbTileset.Items.Count; i++)
    {
```

```
        if (lbTileset.Items[i].ToString() == data.TilesetName)
        {
            MessageBox.Show("Level already contains a tileset with this name.", "Existing
tileset");
            return;
        }
    }

    tileSetData.Add(data);
    tileSets.Add(tileset);

    lbTileset.Items.Add(data.TilesetName);

    pbTilesetPreview.Image = image;
    tileSetImages.Add(image);

    lbTileset.SelectedIndex = lbTileset.Items.Count - 1;
    nudCurrentTile.Value = 0;

    mapLayerToolStripMenuItem.Enabled = true;
}
```

I first create an object of type **OpenFileDialog** to browse for tileset data. Tileset data was stored with a .tdat extension so I set the **Filter** property of **ofDialog** to display just .tdat files. I set the **CheckFileExists** and **CheckPathExists** properties to true. I then capture the result of the **ShowDialog** method. If the result was not the user hitting the OK button I exit out of the method. I then have four local variables needed to create a tileset. The variable **data** will hold the **TilesetData** for the tileset being read in. The **texture** variable will hold the **Texture2D** associated with the tile set. I also have a **Tileset** variable called **tileset** to hold the tileset created and the variable **image** holds the image to display in **pbTilesetPreview** and add to **tileSetImages**.

In a try-catch block I try to deserialize the tileset data. I then try to read in the **Texture2D** for the tileset like I have before. I create a **Stream** to the image location and use the **FromStream** method to try and read the image into a **Texture2D**. I also try and read in the image as a **GDI+** image. If an exception was thrown I display it in a message box and exit the method. Finally I assign **tileset** to be a new **Tileset** using the information read in. If any exceptions were thrown I display them in a message box and then exit the method.

There is next a for loop that will loop through all of the items in the **Items** collection of **lbTileset** to check to see if a tileset with the name of the loaded tileset exists already. If it does I display an error message in a message box and then exit the method.

I then add the **data** variable to **tileSetData** and **tileset** to **tilesets**. I also add the **TilesetName** of the **data** variable to **lbTileset**. I set the **Image** property of **pbTilesetPreview** to the **image** variable and add it to the **tileSetImages** collection. I also set the **SelectedIndex** property of **lbTileset** to be the **Count** property of the **Items** collection minus 1 and set the **Value** property of **nudCurrentTile** to 0 as well. The last step is to set the **Enabled** property of **mapLayerToolStripMenuItem** to true.

The last thing that I'm going to cover in this tutorial is opening a saved map layer. Add the following method to the **Open Menu Item Handler** region.

```
void openLayerToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog ofDialog = new OpenFileDialog();
    ofDialog.Filter = "Map Layer Data (*.mldt)|*.mldt";
    ofDialog.CheckPathExists = true;
    ofDialog.CheckFileExists = true;
```

```
    DialogResult result = ofDialog.ShowDialog();

    if (result != DialogResult.OK)
        return;

    MapLayerData data = null;

    try
    {
        data = XnaSerializer.Deserialize<MapLayerData>(ofDialog.FileName);
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Error reading map layer");
        return;
    }

    for (int i = 0; i < clbLayers.Items.Count; i++)
    {
        if (clbLayers.Items[i].ToString() == data.MapLayerName)
        {
            MessageBox.Show("Layer by that name already exists.", "Existing layer");
            return;
        }
    }

    clbLayers.Items.Add(data.MapLayerName, true);

    layers.Add(MapLayer.FromMapLayerData(data));

    if (map == null)
        map = new TileMap(tileSets, layers);
}
```

This method has the same basic layout as the last. I create an **OpenFileDialog** object to select the **MapLayerData** to be opened. I set the **Filter** property so that it will only display **MapLayerData** files with the extension **mldt**. I set the **CheckPathExists** and **CheckFileExists** properties to true. I display the dialog using the **ShowDialog** method and capture the result. If the result was not the **OK** button I exit the method.

There is then a local variable that will hold a **MapLayerData** object. I try and deserialize the file in a try-catch block. If there was an error deserializing the object I display an error message and exit the method. A for loop follows next that checks to see if there is already a map layer with the same name as the map layer just read in. If there is I display an message box and exit the method. I then add the **MapLayerName** property of the layer to the items of **clbLayers** passing in true so that the item will be checked. I then add a new **MapLayer** using the **FromMapLayerData** method and the **data** variable that was just read in. If the **map** field is null I create a new **TileMap**.

I think I'm going to end this tutorial here. I've added in even more functionality to the level editor. I think the next tutorial will be on the game instead of editors to give you a break from editors. I encourage you to visit the news page of my site, XNA Game Programming Adventures, for the latest news on my tutorials.

Good luck in your game programming adventures!

Jamie McMahon