XNA 4.0 RPG Tutorials

Part 8

Updating Character Generator and Tile Engine

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.

In this tutorial I'm going to be updating the character generator to display a picture of the player's character choice. I will also in the **GamePlayScreen** class load the appropriate image. I'm going to do a little work on the tile engine as well, make it so that you don't always have to subtract the position of the camera and draw only the tiles that need to be drawn.

First thing to change is in the **StartMenuScreen**. I don't want the game to jump straight to the **GamePlayScreen**. if the player decides to load a game. The reason will become apparent shortly. What I did in the **menuItem_Selected** method is comment out pushing the **GamePlayScreen** as the current state if the **loadGame** was selected. Change that method to the following.

```
private void menuItem_Selected(object sender, EventArgs e)
{
    if (sender == startGame)
    {
        StateManager.PushState(GameRef.CharacterGeneratorScreen);
    }

    if (sender == loadGame)
    {
        // StateManager.PushState(GameRef.GamePlayScreen);
    }

    if (sender == exitGame)
    {
        GameRef.Exit();
    }
}
```

I'm now going to update the **CharacterGeneratorScreen**. The first step is to add in a **PictureBox** field for the preview of the character. The other is to add in a 2D array of **Texture2D** for the images. Add these two fields to the **Field** region of the **CharacterGeneratorScreen**.

```
PictureBox characterImage;
Texture2D[,] characterImages;
```

You now want to add the **PictureBox** to the control manager and you want to read in the images. I added a call in the **LoadContent** method to a method I wrote to read in the images, **LoadImages**. I also wired the handler for the **SelectionChanged** event of the **LeftRightSelectors**. Change the **LoadContent** method and **CreateControls** methods to the following. Also, add in the **LoadImages** and **selectionChanged** methods to the **Method** region.

```
protected override void LoadContent()
{
    base.LoadContent();

    LoadImages();
    CreateControls();
}

private void CreateControls()
{
    Texture2D leftTexture = Game.Content.Load<Texture2D>(@"GUI\leftarrowUp");
    Texture2D rightTexture = Game.Content.Load<Texture2D>(@"GUI\rightarrowUp");
    Texture2D stopTexture = Game.Content.Load<Texture2D>(@"GUI\StopBar");

    backgroundImage = new PictureBox(
        Game.Content.Load<Texture2D>(@"Backgrounds\titlescreen"),
        GameRef.ScreenRectangle);
    ControlManager.Add(backgroundImage);

    Label label1 = new Label();

    label1.Text = "Who will search for the Eyes of the Dragon?";
    label1.Size = label1.SpriteFont.MeasureString(label1.Text);
    label1.Position = new Vector2((GameRef.Window.ClientBounds.Width - label1.Size.X) / 2, 150);

    ControlManager.Add(label1);

    genderSelector = new LeftRightSelector(leftTexture, rightTexture, stopTexture);
    genderSelector.SetItems(genderItems, 125);
    genderSelector.Position = new Vector2(label1.Position.X, 200);
    genderSelector.SelectionChanged += new EventHandler(selectionChanged);

    ControlManager.Add(genderSelector);

    classSelector = new LeftRightSelector(leftTexture, rightTexture, stopTexture);
    classSelector.SetItems(classItems, 125);
    classSelector.Position = new Vector2(label1.Position.X, 250);
    classSelector.SelectionChanged += selectionChanged;

    ControlManager.Add(classSelector);

    LinkLabel linkLabel1 = new LinkLabel();
    linkLabel1.Text = "Accept this character.";
    linkLabel1.Position = new Vector2(label1.Position.X, 300);
    linkLabel1.Selected += new EventHandler(linkLabel1_Selected);

    ControlManager.Add(linkLabel1);

    characterImage = new PictureBox(
        characterImages[0, 0],
        new Rectangle(500, 200, 96, 96),
        new Rectangle(0, 0, 32, 32));
    ControlManager.Add(characterImage);

    ControlManager.NextControl();
}

private void LoadImages()
{
    characterImages = new Texture2D[genderItems.Length, classItems.Length];

    for (int i = 0; i < genderItems.Length; i++)
    {
        for (int j = 0; j < classItems.Length; j++)
        {
            characterImages[i, j] = Game.Content.Load<Texture2D>(@"PlayerSprites\" +
genderItems[i] + classItems[j]);
        }
    }
}
```

```
void selectionChanged(object sender, EventArgs e)
{
    characterImage.Image = characterImages[genderSelector.SelectedIndex,
classSelector.SelectedIndex];
}
```

The **LoadContent** method calls the **LoadImages** method before **CreateControls**. It does this becasue in the **CreateControls** method I use the images loaded in for the **PictureBox** to display the preview of the player's character. In the **CreateControls** wires the handler for the **SelectionChanged** event for the **genderSelector** and **classSelector** to the **selectionChanged** method. It creates the **PictureBox** passing in the texture for a male fighter, the default character. I chose an arbitrary destination rectangle for the **PictureBox** and for the source rectangle I used the first frame of the animation. The **PictureBox** is then added to the **ControlManager**.

The **LoadImages** method is where I load in the sprite sheets for the different types of characters. The first step is to create an array that holds the images. For the first dimension I use the length of the **genderItems** array. For the second the length of the **classItems** array. The there is a set of nested for loops. The outer loop is for the gender and the inner loop for the class. To find the name of images I take the folder where the sprites are located add the gender from the **genderItems** array and then the class from the **classItems** array. This works because I named the sprites **gender** + **class**.

The **selectionChanged** event is where I change the image based on what the choices are in the left and right selectors. I use the **SelectedIndex** properties of the selectors for the index of each dimension. The **genderSelector** for the first dimension and **classSelector** for the second.

The next problem is getting the player's selection to the **GamePlayScreen**. I'm not ready to add in a class for the world yet. There is much I want to add before I get there. What I elected to do was add in properties to the **CharacterGeneratorScreen** that return the **SelectedItem** property of the gender and the class selectors. Then, in the **GamePlayScreen**, I use the properties to retrieve the values. First, to the **Property** region of the **CharacterGeneratorScreen** add in the following properties.

```
public string SelectedGender
{
    get { return genderSelector.SelectedItem; }
}

public string SelectedClass
{
    get { return classSelector.SelectedItem; }
}
```

Now, in the **LoadContent** method of the **GamePlayScreen** you use the values of these properties to load in the appropriate sprite sheet. This is why I removed going straight to the **GamePlayScreen** from the **StartMenuScreen**. If you do that you will get a null reference exception because the **LoadContent** method of the **CharacterGeneratorScreen** has not been called. It won't be called if you don't use the screen manager to push it on the stack or change to that screen. Change the **LoadContent** method of the **GamePlayScreen** to the following.

```
protected override void LoadContent()
{
    Texture2D spriteSheet = Game.Content.Load<Texture2D>(
        @"PlayerSprites\" +
        GameRef.CharacterGeneratorScreen.SelectedGender +
        GameRef.CharacterGeneratorScreen.SelectedClass);
```

```
    Dictionary<AnimationKey, Animation> animations = new Dictionary<AnimationKey, Animation>();

    Animation animation = new Animation(3, 32, 32, 0, 0);
    animations.Add(AnimationKey.Down, animation);

    animation = new Animation(3, 32, 32, 0, 32);
    animations.Add(AnimationKey.Left, animation);

    animation = new Animation(3, 32, 32, 0, 64);
    animations.Add(AnimationKey.Right, animation);

    animation = new Animation(3, 32, 32, 0, 96);
    animations.Add(AnimationKey.Up, animation);

    sprite = new AnimatedSprite(spriteSheet, animations);

    base.LoadContent();

    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
    Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");
    Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    List<Tileset> tilesets = new List<Tileset>();
    tilesets.Add(tileset1);
    tilesets.Add(tileset2);

    MapLayer layer = new MapLayer(40, 40);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);

            layer.SetTile(x, y, tile);
        }
    }

    MapLayer splatter = new MapLayer(40, 40);

    Random random = new Random();

    for (int i = 0; i < 80; i++)
    {
        int x = random.Next(0, 40);
        int y = random.Next(0, 40);
        int index = random.Next(2, 14);

        Tile tile = new Tile(index, 0);
        splatter.SetTile(x, y, tile);
    }

    splatter.SetTile(1, 0, new Tile(0, 1));
    splatter.SetTile(2, 0, new Tile(2, 1));
    splatter.SetTile(3, 0, new Tile(0, 1));

    List<MapLayer> mapLayers = new List<MapLayer>();
    mapLayers.Add(layer);
    mapLayers.Add(splatter);

    map = new TileMap(tilesets, mapLayers);
}
```

What I want to do next is to do a little work on the tile engine, mostly the **Camera** class. I'm going to implement being able to zoom in and zoom out to the **Camera** class. I'm also going to make it so that

you are not constantly having to subtract the position of the camera in your drawing code. So, open the code for your **Camera** class. The first step is to check for keys or buttons to make the camera zoom in or out and then call methods to have the camera zoom in or out. Change the **Update** method to the following and add the methods **ZoomIn** and **ZoomOut**.

```
public void Update(GameTime gameTime)
{
    if (InputHandler.KeyReleased(Keys.PageUp) ||
        InputHandler.ButtonReleased(Buttons.LeftShoulder, PlayerIndex.One))
        ZoomIn();
    else if (InputHandler.KeyReleased(Keys.PageDown) ||
        InputHandler.ButtonReleased(Buttons.RightShoulder, PlayerIndex.One))
        ZoomOut();

    if (mode == CameraMode.Follow)
        return;

    Vector2 motion = Vector2.Zero;

    if (InputHandler.KeyDown(Keys.Left) ||
        InputHandler.ButtonDown(Buttons.RightThumbstickLeft, PlayerIndex.One))
        motion.X = -speed;
    else if (InputHandler.KeyDown(Keys.Right) ||
        InputHandler.ButtonDown(Buttons.RightThumbstickRight, PlayerIndex.One))
        motion.X = speed;

    if (InputHandler.KeyDown(Keys.Up) ||
        InputHandler.ButtonDown(Buttons.RightThumbstickUp, PlayerIndex.One))
        motion.Y = -speed;
    else if (InputHandler.KeyDown(Keys.Down) ||
        InputHandler.ButtonDown(Buttons.RightThumbstickDown, PlayerIndex.One))
        motion.Y = speed;

    if (motion != Vector2.Zero)
    {
        motion.Normalize();
        position += motion * speed;
        LockCamera();
    }
}

private void ZoomIn()
{
    zoom += .25f;

    if (zoom > 2.5f)
        zoom = 2.5f;
}

private void ZoomOut()
{
    zoom -= .25f;

    if (zoom < .5f)
        zoom = .5f;
}
```

I check to see if the **Page Up** or **Left Shoulder** on the game pad are pressed. If they are I call the method **ZoomIn**. I then check to see if the **Page Down** or **Right Shoulder** on the game pad are pressed. If they are I call the method **ZoomOut**.

Zooming in is the process of making things larger and the **ZoomIn** method handles that. To do that I increment the **zoom** field by .25 or 25 percent. I see the **zoom** field is greater than 2.5, or 250 percent

larger than normal. If it is I set it to be 2.5 as I think that is more than large enough. Zooming out is the process of making things smaller and the method **ZoomOut** handles doing that. To do that I decrement the **zoom** field by .25. I see if it is smaller that .5 and if it is set its value to .5 or 50 percent smaller.

Our map still doesn't zoom in or out though. I will get to that in a minute. In the **GamePlayScreen** where we do the call to **Begin** of the **SpriteBatch** we are using the identity matrix for our transformation matrix. You want to instead use a transformation matrix related to the camera. You will want to scale the map according to the **zoom** field and than you will want to translate, or move, using the position of the camera. You combine transformations by multiplying the matrix for each together. The order in which you do this is important. You should following the following rule: Identity, Scale, Rotation, Orbit, and Translate. So, we want to add in two transformation matricies. The first to scale the map according to the **zoom** field and the second to translate the map using the **Position** of the camera. The **Matrix** class has static methods for performing these operations. It will also be helpful to know the view port the camera so I added in a property to return the **viewportRectangle** field. Add the following properties to the **camera** class in the **Property** region.

```
public Matrix Transformation
{
    get { return Matrix.CreateScale(zoom) *
        Matrix.CreateTranslation(new Vector3(-Position, 0f)); }
}

public Rectangle ViewportRectangle
{
    get { return new Rectangle(
        viewportRectangle.X,
        viewportRectangle.Y,
        viewportRectangle.Width,
        viewportRectangle.Height); }
}
```

For the **CreateScale** method call I just pass in the **zoom** field. For the **CreateTranslation** method call I pass in a **Vector3**. To create it I use negative position and 0f for the other. I use negative position because we subtract the camera's position. I use 0f, the **Z** coordinate because we ignore the **Z** coordinate because we are working in two dimensions, not three.

Back in the **Draw** method of the **GamePlayScreen** you will want to switch the **Matrix.Identity** with the transformation matrix of the **Camera** class. Update the **Draw** method of the **GamePlayScreen** to the following.

```
public override void Draw(GameTime gameTime)
{
    GameRef.SpriteBatch.Begin(
        SpriteSortMode.Deferred,
        BlendState.AlphaBlend,
        SamplerState.PointClamp,
        null,
        null,
        null,
        player.Camera.Transformation);

    map.Draw(GameRef.SpriteBatch, player.Camera);
    sprite.Draw(gameTime, GameRef.SpriteBatch, player.Camera);

    base.Draw(gameTime);

    GameRef.SpriteBatch.End();
}
```

I also set the **SpriteSortMode** to **Deferred** which gives us a bit of a performance boost. You will now have to go to the **TileMap** class and remove the code that was subtracting the camera's position when finding the destination rectangles. Change **Draw** method of the **TileMap** class to the following.

```
public void Draw(SpriteBatch spriteBatch, Camera camera)
{
    Rectangle destination = new Rectangle(0, 0, Engine.TileWidth, Engine.TileHeight);
    Tile tile;

    foreach (MapLayer layer in mapLayers)
    {
        for (int y = 0; y < layer.Height; y++)
        {
            destination.Y = y * Engine.TileHeight;

            for (int x = 0; x < layer.Width; x++)
            {
                tile = layer.GetTile(x, y);

                if (tile.TileIndex == -1 || tile.Tileset == -1)
                    continue;

                destination.X = x * Engine.TileWidth;

                spriteBatch.Draw(
                    tilesets[tile.Tileset].Texture,
                    destination,
                    tilesets[tile.Tileset].SourceRectangles[tile.TileIndex],
                    Color.White);
            }
        }
    }
}
```

You also need to update the **Draw** method of the **AnimatedSprtie** class so you are no longer subtracting the position of the camera. Change that method to the following.

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera)
{
    spriteBatch.Draw(
        texture,
        position,
        animations[currentAnimation].CurrentFrameRect,
        Color.White);
}
```

So now the game builds and runs like before. The zooming of the camera doesn't behave all that nicely though. For one thing, with the map scaled it is not locking right. If you have a high zoom you can't see the far edges of the map. Also, with a small zoom you will see the blue background. That is because of the size of the map. Making a larger map will stop that from happening. Another thing is if you zoom the camera should move with the zoom. To allow the camera to see more of the map you need to change the **LockCamera** method. If the zoom level reduces the size of the map the **WidthInPixels** and **HeightInPixels** properties are smaller. If the zoom level increases the size of the map they increase as well. To lock the camera properly you can multiply these values by the **zoom** field. Change the **LockCamera** method to the following.

```
private void LockCamera()
{
    position.X = MathHelper.Clamp(position.X,
        0,
        TileMap.WidthInPixels * zoom - viewportRectangle.Width);
    position.Y = MathHelper.Clamp(position.Y,
```

```
            0,
            TileMap.HeightInPixels * zoom - viewportRectangle.Height);
}
```

Now, if you zoom in you can see the entire map. Moving the camera's position when you zoom in or out is a little trickier. What I did is after modifying the **zoom** field is create a **Vector2** multiplying **Position** by **zoom** to scale the position. I then called a method I wrote, **SnapToPosition**. That will snap the camera to a **Vector2**. Change the **ZoomIn** and **ZoomOut** method to the following. Also add the **SnapToPosition** method.

```
public void ZoomIn()
{
    zoom += .25f;

    if (zoom > 2.5f)
        zoom = 2.5f;

    Vector2 newPosition = Position * zoom;
    SnapToPosition(newPosition);
}

public void ZoomOut()
{
    zoom -= .25f;

    if (zoom < .5f)
        zoom = .5f;

    Vector2 newPosition = Position * zoom;
    SnapToPosition(newPosition);
}

private void SnapToPosition(Vector2 newPosition)
{
    position.X = newPosition.X - viewportRectangle.Width / 2;
    position.Y = newPosition.Y - viewportRectangle.Height / 2;
    LockCamera();
}
```

The **SnapToPosition** method sets the **X** value of the position to the new position of the camera minus half the width of the view port. Similarly, the **Y** value is set to the new position of the camera minus half the height of the view port. Finally, I call the **LockCamera** method to keep it from scrolling off the edges. This has the camera working a little funny if it is in follow mode when zooming in. It is rather jumpy. The solution I found was moving the code for checking if the player wants to zoom the camera outside of the camera class in to the **GamePlayScreen**. This way if the camera is in follow mode you can call the **LockToSprite** method. Speaking of which, you need to update that method to have it work properly as well. What you do is similar to what you did in the **LockCamera** method. You multiply the sprite's position plus have the width or height by the **zoom** field. Change the **Update** and **LockToSprite** methods in the camera class to the following.

```
public void Update(GameTime gameTime)
{
    if (mode == CameraMode.Follow)
        return;

    Vector2 motion = Vector2.Zero;

    if (InputHandler.KeyDown(Keys.Left) ||
        InputHandler.ButtonDown(Buttons.RightThumbstickLeft, PlayerIndex.One))
        motion.X = -speed;
    else if (InputHandler.KeyDown(Keys.Right) ||
        InputHandler.ButtonDown(Buttons.RightThumbstickRight, PlayerIndex.One))
```

```
        motion.X = speed;

    if (InputHandler.KeyDown(Keys.Up) ||
        InputHandler.ButtonDown(Buttons.RightThumbstickUp, PlayerIndex.One))
        motion.Y = -speed;
    else if (InputHandler.KeyDown(Keys.Down) ||
        InputHandler.ButtonDown(Buttons.RightThumbstickDown, PlayerIndex.One))
        motion.Y = speed;

    if (motion != Vector2.Zero)
    {
        motion.Normalize();
        position += motion * speed;
        LockCamera();
    }

}

public void LockToSprite(AnimatedSprite sprite)
{
    position.X = (sprite.Position.X + sprite.Width / 2) * zoom
                 - (viewportRectangle.Width / 2);
    position.Y = (sprite.Position.Y + sprite.Height / 2) * zoom
                 - (viewportRectangle.Height / 2);
    LockCamera();
}
```

Now I'm going to update the **Update** method of the **GamePlayScreen**. I moved the code from the **Camera** class to control zooming in and out. I also added a check that if the player's camera is in follow mode to lock the camera to the player's sprite. Change the **Update** method to the following.

```
public override void Update(GameTime gameTime)
{
    player.Update(gameTime);
    sprite.Update(gameTime);

    if (InputHandler.KeyReleased(Keys.PageUp) ||
        InputHandler.ButtonReleased(Buttons.LeftShoulder, PlayerIndex.One))
    {
        player.Camera.ZoomIn();
        if (player.Camera.CameraMode == CameraMode.Follow)
            player.Camera.LockToSprite(sprite);
    }
    else if (InputHandler.KeyReleased(Keys.PageDown) ||
        InputHandler.ButtonReleased(Buttons.RightShoulder, PlayerIndex.One))
    {
        player.Camera.ZoomOut();
        if (player.Camera.CameraMode == CameraMode.Follow)
            player.Camera.LockToSprite(sprite);
    }

    Vector2 motion = new Vector2();

    if (InputHandler.KeyDown(Keys.W) ||
        InputHandler.ButtonDown(Buttons.LeftThumbstickUp, PlayerIndex.One))
    {
        sprite.CurrentAnimation = AnimationKey.Up;
        motion.Y = -1;
    }
    else if (InputHandler.KeyDown(Keys.S) ||
        InputHandler.ButtonDown(Buttons.LeftThumbstickDown, PlayerIndex.One))
    {
        sprite.CurrentAnimation = AnimationKey.Down;
        motion.Y = 1;
    }

    if (InputHandler.KeyDown(Keys.A) ||
        InputHandler.ButtonDown(Buttons.LeftThumbstickLeft, PlayerIndex.One))
```

```
    {
        sprite.CurrentAnimation = AnimationKey.Left;
        motion.X = -1;
    }
    else if (InputHandler.KeyDown(Keys.D) ||
        InputHandler.ButtonDown(Buttons.LeftThumbstickRight, PlayerIndex.One))
    {
        sprite.CurrentAnimation = AnimationKey.Right;
        motion.X = 1;
    }

    if (motion != Vector2.Zero)
    {
        sprite.IsAnimating = true;
        motion.Normalize();

        sprite.Position += motion * sprite.Speed;
        sprite.LockToMap();

        if (player.Camera.CameraMode == CameraMode.Follow)
            player.Camera.LockToSprite(sprite);
    }
    else
    {
        sprite.IsAnimating = false;
    }

    if (InputHandler.KeyReleased(Keys.F) ||
        InputHandler.ButtonReleased(Buttons.RightStick, PlayerIndex.One))
    {
        player.Camera.ToggleCameraMode();
        if (player.Camera.CameraMode == CameraMode.Follow)
            player.Camera.LockToSprite(sprite);
    }

    if (player.Camera.CameraMode != CameraMode.Follow)
    {
        if (InputHandler.KeyReleased(Keys.C) ||
            InputHandler.ButtonReleased(Buttons.LeftStick, PlayerIndex.One))
        {
            player.Camera.LockToSprite(sprite);
        }
    }

    base.Update(gameTime);
}
```

While we have this class open, let's make the map a little bigger. I had used 40 for the width and the height of the map. I changed my **LoadContent** method so that the map is now 100 by 100 tiles. Change the **LoadContent** method to the following.

```
protected override void LoadContent()
{
    Texture2D spriteSheet = Game.Content.Load<Texture2D>(
        @"PlayerSprites\" +
        GameRef.CharacterGeneratorScreen.SelectedGender +
        GameRef.CharacterGeneratorScreen.SelectedClass);

    Dictionary<AnimationKey, Animation> animations = new Dictionary<AnimationKey, Animation>();

    Animation animation = new Animation(3, 32, 32, 0, 0);
    animations.Add(AnimationKey.Down, animation);

    animation = new Animation(3, 32, 32, 0, 32);
    animations.Add(AnimationKey.Left, animation);

    animation = new Animation(3, 32, 32, 0, 64);
    animations.Add(AnimationKey.Right, animation);
```

```
    animation = new Animation(3, 32, 32, 0, 96);
    animations.Add(AnimationKey.Up, animation);

    sprite = new AnimatedSprite(spriteSheet, animations);

    base.LoadContent();

    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
    Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");
    Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    List<Tileset> tilesets = new List<Tileset>();
    tilesets.Add(tileset1);
    tilesets.Add(tileset2);

    MapLayer layer = new MapLayer(100, 100);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);

            layer.SetTile(x, y, tile);
        }
    }

    MapLayer splatter = new MapLayer(100, 100);

    Random random = new Random();

    for (int i = 0; i < 100; i++)
    {
        int x = random.Next(0, 100);
        int y = random.Next(0, 100);
        int index = random.Next(2, 14);

        Tile tile = new Tile(index, 0);
        splatter.SetTile(x, y, tile);
    }

    splatter.SetTile(1, 0, new Tile(0, 1));
    splatter.SetTile(2, 0, new Tile(2, 1));
    splatter.SetTile(3, 0, new Tile(0, 1));

    List<MapLayer> mapLayers = new List<MapLayer>();
    mapLayers.Add(layer);
    mapLayers.Add(splatter);

    map = new TileMap(tilesets, mapLayers);
}
```

Things are coming together but there is still more work to be done. I think that is more than enough for this tutorial. I'd like to try and keep them to a reasonable length so that you don't have too much to digest at once. 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