Creating a Role Playing Game with XNA Game Studio 3.0
Part 22
Updating Tile Engine

To follow along with this tutorial you will have to have read the previous tutorials to understand much of what it going on. You can find a list of tutorials here: XNA 3.0 Role Playing Game Tutorials You will also find the latest version of the project on the web site on that page. If you want to follow along and type in the code from this PDF as you go, you can find the previous project at this link: Eyes of the Dragon - Version 21 You can download the graphics from this link: Graphics.zip

I have been wanting to do this for some time now. The tile engine works okay but I want to control scrolling the map in the **Game1** class instead of the **TileEngine** class. There are a number of reasons for this. The most important being that once I get to incorporating game play into the game I will need easy access to where the player is on the map. I will also be making an optimization to the rendering of the map. At the moment it only draws what is visible but by making a small change it will make the rendering process a little faster.

To get started I am going to remove the **Camera** object from the **TileEngine** class along with the logic that scrolls the map and place it in the **Game1** class. The **Camera** object is static and private. There is also a get only property to get the camera object. This is the field and property.

```
static Camera camera = new Camera();

public static Camera Camera
{
    get { return camera; }
}
```

For the code controlling the logic of the camera will go inside the **HandleActionScreenInput** method. I just copied the code out **Update** method of the **TileEngine** and pasted it into that method. I also made a minor change to that method. I used if-else statements to control the flow of input. So if the player presses V to view the character scrolling the map will be disabled. The is the code for the new **HandleActionScreenInput** method.

```
private void HandleActionScreeenInput()
{
    if (CheckKey(Keys.Escape))
    {
        this.Exit();
    }
    else if (CheckKey(Keys.V))
    {
        activeScreen.Enabled = false;
        activeScreen = viewCharacterScreen;
        viewCharacterScreen.SetPlayerCharacter(playerCharacter);
        activeScreen.Show();
    }
    else
    {
        Vector2 motion = new Vector2();
```

```
            if (newState.IsKeyDown(Keys.Up) || newState.IsKeyDown(Keys.NumPad8))
            {
                motion.Y--;
            }

            if (newState.IsKeyDown(Keys.Right) || newState.IsKeyDown(Keys.NumPad6))
            {
                motion.X++;
            }

            if (newState.IsKeyDown(Keys.Down) || newState.IsKeyDown(Keys.NumPad2))
            {
                motion.Y++;
            }

            if (newState.IsKeyDown(Keys.Left) || newState.IsKeyDown(Keys.NumPad4))
            {
                motion.X--;
            }

            if (newState.IsKeyDown(Keys.NumPad9))
            {
                motion.X++;
                motion.Y--;
            }

            if (newState.IsKeyDown(Keys.NumPad3))
            {
                motion.X++;
                motion.Y++;
            }

            if (newState.IsKeyDown(Keys.NumPad1))
            {
                motion.X--;
                motion.Y++;
            }

            if (newState.IsKeyDown(Keys.NumPad7))
            {
                motion.X--;
                motion.Y--;
            }

            if (motion != Vector2.Zero)
            {
                motion.Normalize();
            }

            camera.Position += motion * camera.Speed;

            camera.LockCamera();
        }
    }
```

I made many changes to the **TileEngine** class. It is now a static class that has a few fields and properties related to the **TileEngine**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;

namespace New2DRPG
{
    public static class TileEngine
    {
        static int tileWidth = 64;
        static int tileHeight = 48;
        static int viewPortWidth = 1024;
        static int viewPortHeight = 720;

        public static int TileWidth
        {
            get { return tileWidth; }
        }

        public static int TileHeight
        {
            get { return tileHeight; }
        }

        public static Vector2 ViewPortVector
        {
            get
            {
                return new Vector2(viewPortWidth + tileWidth,
                    viewPortHeight + tileHeight);
            }
        }

        public static int ViewPortWidth
        {
            get { return viewPortWidth; }
        }

        public static int ViewPortHeight
        {
            get { return viewPortHeight; }
        }

    }
}
```

As you can see there are four fields and four get only properties for those fields. The fields are **tileWidth**, **tileHeight**, **viewPortWidth** and **viewPortHeight**. The properties for those field, in the same order are **TileWidth**, **TileHeight**, **ViewPortWidth** and **ViewPortHeight**. **tileWidth** and and **tileHeight** are for the width and height of the tiles on the screen. **viewPortWidth** and **viewPortHeight** are for the height and width of the view port.

I did a rewrite of the **TileMap** and **TileMapLayer** classes as well. The **TileMap** will now ask the layers to draw themselves instead of drawing them. I will start with the **TileMap** class. This is the code for the **TileMap** class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;


namespace New2DRPG
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class TileMap : Microsoft.Xna.Framework.DrawableGameComponent
    {
        List<TileMapLayer> tileMapLayers = new List<TileMapLayer>();

        Random random = new Random();

        SpriteBatch spriteBatch;
        ContentManager Content;

        int mapWidth;
        int mapHeight;

        static int widthInPixels;
        static int heightInPixels;

        Tileset tileset;
        Texture2D texture;

        public TileMap(int tileMapWidth, int tileMapHeight, Tileset tileset,
                       Game game)
            : base(game)
        {
            spriteBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
            Content =
                (ContentManager)Game.Services.GetService(typeof(ContentManager));

            this.tileset = tileset;

            LoadContent();

            mapWidth = tileMapWidth;
            mapHeight = tileMapHeight;

            TileMapLayer layer = new TileMapLayer(tileMapWidth, tileMapHeight);

            for (int x = 0; x < tileMapWidth; x++)
                for (int y = 0; y < tileMapHeight; y++)
                    layer.SetTile(x, y, 0);

            tileMapLayers.Add(layer);
            layer = new TileMapLayer(mapWidth, mapHeight);

            for (int x = 0; x < tileMapWidth; x++)
                for (int y = 0; y < tileMapHeight; y++)
                {
                    layer.SetTile(x, y, -1);
                    if (random.Next(0, 50) < 5)
                    {
                        layer.SetTile(x, y, tileset.Tiles.Count - 1);
                    }
                }

            tileMapLayers.Add(layer);
        }

        public static int WidthInPixels
        {
            get { return widthInPixels; }
        }

        public static int HeightInPixels
        {
            get { return heightInPixels; }
        }

        protected override void LoadContent()
        {
            texture = Content.Load<Texture2D>(@"TileSets\" + tileset.TextureName);
            base.LoadContent();
        }

        public override void Initialize()
        {
            base.Initialize();
        }

        public override void Update(GameTime gameTime)
        {
            widthInPixels = mapWidth * TileEngine.TileWidth;
            heightInPixels = mapHeight * TileEngine.TileHeight;

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            foreach (TileMapLayer layer in tileMapLayers)
            {
                layer.Draw(spriteBatch, texture, tileset);
            }
            base.Draw(gameTime);
        }

        public void Hide()
        {
            Visible = false;
            Enabled = false;
        }
        public void Show()
        {
            Visible = true;
            Enabled = true;
        }
    }
}
```

I added in many new fields to this class. There is a **ContentManager** object, **Content**, that will be used to load in the texture for the **Tileset** object. I will need the **Tileset** object so there is a field for that, **tileset**. I also need the **Texture2D** for the tile set so there is a **Texture2D** object called **texture**. There are four integer variables as well, two of them are static. The first two hold the width and height of the map. They are **mapWidth** and **mapHeight**. The two static fields hold the width and height of the map in pixels. They are **widthInPixels** and **heightInPixels**. There are two get only properties to expose the fields called **WidthInPixels** and **HeightInPixels**. There are also the original three fields. One for the **SpriteBatch** object, **spriteBatch**. There is a **Random** object called **random**. The last field is a **List<TileMapLayers>**, **tileMapLayers**.

The constuctor has four parameters. The first one **tileMapWidth** is of course the width of the map and **tileMapHeight** is the height of the map. The third one **tileset** is used to pass in the **Tileset** object. The last parameter **game** is for the **Game** object. In the constructor I get the **SpriteBatch** object and **ContentManager** object from the objects that were added to the list of **Game Services**. I set the **tileset** field because I need the name of the texture to load in the **LoadContent** method to load in the texture of the tile set. I call the **LoadContent** method to load in the **Tileset** object for the map. Then I set the **mapWidth** and **mapHeight** fields. Finally I create two layers with **mapWidth** and **mapHeight**. The first layer is filled with the grass tile which is index 0. The second layer is a layer that is mostly empty. There is a small chance that the tile will have the last tile in the tile set which has a transparent background with three rocks in it.

After the two public static get only properties there is the **LoadContent** method. This method just loads in the texture for the tile set. I did nothing new in the **Initialize** method. In the **Update** method I just set the **widthInPixels** to the **mapWidth** field time the width of the tiles using the **TileWidth** property of the **TileEngine** class. I also set the **heightInPixels** variable using a similar method.

The **Draw** method is completely different from the last version. As I mentioned easlier the rendering of the layers will be done in the **TileMapLayer** class. In a foreach loop I loop through all of the layers in the map. I call the **Draw** method of the **TileMapLayer** class. It takes as a parameters a **SpriteBatch** object, the **Texture2D** for the tile set and a **Tileset** object.

There are two other methods in this class **Hide** and **Show**. As their names imply the **Hide** method is used to hide the component and the **Show** method is used to show the component.

Now I will turn my attention to the **TileMapLayer** class. This class didn't change that much I added in two using statements and two new methods. This is the code for the new **TileMapLayer** class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace New2DRPG
{
    class TileMapLayer
    {
        int[,] map;

        public TileMapLayer(int x, int y)
        {
            map = new int[y, x];
        }

        public int TileMapWidth
        {
            get { return map.GetLength(1); }
        }

        public int TileMapHeight
        {
            get { return map.GetLength(0); }
        }

        public void SetTile(int x, int y, int tileIndex)
        {
            map[y, x] = tileIndex;
        }

        public int GetTile(int x, int y)
        {
            return map[y, x];
        }

        private Point VectorToCell(Vector2 vector)
        {
            return new Point(
                        (int)(vector.X / TileEngine.TileWidth),
                        (int)(vector.Y / TileEngine.TileHeight));
        }

        public void Draw(SpriteBatch sBatch, Texture2D texture, Tileset tileset)
        {
            Point cameraPoint = VectorToCell(Game1.Camera.Position);
            Point viewPoint = VectorToCell(Game1.Camera.Position +
                                TileEngine.ViewPortVector);
            Point min = new Point();
            Point max = new Point();
            min.X = cameraPoint.X;
            min.Y = cameraPoint.Y;
            max.X = (int)Math.Min(viewPoint.X, map.GetLength(1));
            max.Y = (int)Math.Min(viewPoint.Y, map.GetLength(0));

            Rectangle tileRectangle = new Rectangle(
                    0,
                    0,
                    TileEngine.TileWidth,
                    TileEngine.TileHeight);
            for (int x = min.X; x < max.X; x++)
            {
                for (int y = min.Y; y < max.Y; y++)
                {
                    if (map[y, x] != -1)
                    {
                        tileRectangle.X = x * TileEngine.TileWidth -
                            (int)Game1.Camera.Position.X;
                        tileRectangle.Y = y * TileEngine.TileHeight -
                            (int)Game1.Camera.Position.Y;

                        sBatch.Draw(texture,
                            tileRectangle,
                            tileset.Tiles[map[y, x]],
                            Color.White);
                    }
                }
            }
        }
    }
}
```

There are now using statements for the **XNA Framework** and the **Graphics** namespace. The first method is from the old **TileMap** class **VectorToCell**. This method takes a **Vector2** and returns a **Point** that corrospondes to the **X** and **Y** coordinates in the map.

The other new method is the **Draw** method that I mentioned previously when I was talking about the **TileMap** class. This method is a lot like the old one. Like in the **Draw** method from the old **TileMap** class it finds the range of tiles to draw. That is specifically the number of tiles that will fit across the screen plus one and the number of tiles that will fit down the screen plus one. Now is where I made a small optimizatoin. Before each time through the nested loops I was creating a new instance of the **Rectangle** class. That is actually a waste as I only need to create one and use it as only the **X** and **Y** values are changing. Constantly creating objects if you don't need to is inefficient. There is the extra instructions involved in creating the object and disposing of the object. As well, there is no need in assigning the **X** and **Y** values if the tile is not visible. I had already done that small optimization earlier when I added the second layer to the **TileMap**. To find the **X** coordinate of the tile I take the **x** value of the loop and multiply it by the **TileWidth** property of the **TileEngine** class and subtracting the position of the **Camera** object in the **Game1**class. I do something similar for the **Y** coordinate. I take the **y** value of the loop, multiply it by the **TileHeight** property of the **TileEngine** class and subtract the position of the camera.

Because of the changes that I have made I needed to make an update to the **ItemSprite** class. The **TileEngine** class no longer has a **Camera** object in it. I needed to update the **ItemSprite** class to use the **Camera** object in the **Game1** class. All that I had to do was change **TileEngine** to **Game1** to get the **Camera** position to solve the problem. This is the new class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using New2DRPG.CoreComponents;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework;

namespace New2DRPG.SpriteClasses
```

```
{
    class ItemSprite : Sprite
    {
        public ItemSprite(Game game, Texture2D texture, Vector2 position)
            : base(game, texture)
        {
            this.position = position;
        }

        public override void Draw(GameTime gameTime)
        {
            spriteBatch.Draw(texture,
                new Rectangle((int)position.X * TileEngine.TileWidth
                        - (int)Game1.Camera.Position.X,
                    (int)position.Y * TileEngine.TileHeight
                        - (int)Game1.Camera.Position.Y,
                    TileEngine.TileWidth,
                    TileEngine.TileHeight),
                    Color.White);

            base.Draw(gameTime);
        }

    }
}
```

I had to make two small changes to the **ActionScreen** class. The reason is the **TileEngine** no longer used for rendering the map. I had to remove adding a **TileEngine** component to the list of game components to adding in a **TileMap** component. So find the **TileEngine** field and remove it. Finally add this field and change the constructor to the following.

```
TileMap tileMap;

public ActionScreen(Game game, SpriteFont gameFont, string tilesetName)
    : base(game)
{
    playerCharacter = new PlayerCharacter(game);
    this.gameFont = gameFont;

    this.tilesetName = tilesetName;
    LoadContent();
    tileMap = new TileMap(50, 50, tileset, game);
    Components.Add(tileMap);
    tileMap.Show();

    viewportWidth = TileEngine.ViewPortWidth;
    viewportHeight = TileEngine.ViewPortHeight;
    screenWidth = game.Window.ClientBounds.Width;
    screenHeight = game.Window.ClientBounds.Height;
}
```

Well that is it for this tutorial. I am already working on coding the next part of Eyes of the Dragon so I encourage you to keep either visiting my site http://xna.jtmbooks.com or my blog, http://xna-rpg.blogspot.com for the latest news on these tutorials.