XNA 4.0 RPG Tutorials

Part 12

Updating Game

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.

This tutorial is going to focus on moving a few things around and update the **World** class. For example, I want to move the player's sprite to the **Player** class. I also want to make a few modifications to the tile engine. To get started load up your solution from last time. Right click the **EyesOfTheDragon** project in the solution explorer and select **Set As StartUp Project**.

'm also going to add a class to go with the **World** class called **Level**. The **World** class will be made up of levels. Each level is an area of the game. The **Level** will have various things associated with it, including a **TileMap**. After adding the **Level** class I will integrate it into the **World** class. I will then move to using the **World** class in the **GamePlayScreen** instead of a **TileMap** for drawing the world.

Right click the **WorldClassses** folder in the **XRpgLibrary** project, select **Add** and then **Class**. Name this new class **Level**. This is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using XRpgLibrary.TileEngine;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace XRpgLibrary.WorldClasses
{
    public class Level
    {
        #region Field Region

        readonly TileMap map;

        #endregion

        #region Property Region

        public TileMap Map
        {
            get { return map; }
        }

        #endregion

        #region Constructor Region
```

```
        public Level(TileMap tileMap)
        {
            map = tileMap;
        }

        #endregion

        #region Method Region

        public void Update(GameTime gameTime)
        {
        }

        public void Draw(SpriteBatch spiteBatch, Camera camera)
        {
            map.Draw(spiteBatch, camera);
        }

        #endregion
    }
}
```

This is a rather simple class but it will be fleshed out more as the game progresses. There is a readonly **TileMap** field called **map**. I set it readonly so it can't accidentally get assigned to. There is a public property that exposes the **map** field called **Map** that is get only. The constructor takes a **TileMap** as a parameter and sets the **map** field using it. There is an **Update** method to update the **Level**. There is also a **Draw** method to draw the map.

I updated the **World** class to use the **Level** class. I also made it a **DrawableGameComponent**. Change the **World** class to the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

using RpgLibrary.CharacterClasses;
using RpgLibrary.ItemClasses;

using XRpgLibrary.TileEngine;
using XRpgLibrary.SpriteClasses;

namespace XRpgLibrary.WorldClasses
{
    public class World : DrawableGameComponent
    {
        #region Graphic Field and Property Region

        Rectangle screenRect;

        public Rectangle ScreenRectangle
        {
            get { return screenRect; }
        }

        #endregion

        #region Item Field and Property Region

        ItemManager itemManager = new ItemManager();

        #endregion
```

```
        #region Level Field and Property Region

        readonly List<Level> levels = new List<Level>();
        int currentLevel = -1;

        public List<Level> Levels
        {
            get { return levels; }
        }

        public int CurrentLevel
        {
            get { return currentLevel; }
            set
            {
                if (value < 0 || value >= levels.Count)
                    throw new IndexOutOfRangeException();

                if (levels[value] == null)
                    throw new NullReferenceException();

                currentLevel = value;
            }
        }

        #endregion

        #region Constructor Region

        public World(Game game, Rectangle screenRectangle)
            : base(game)
        {
            screenRect = screenRectangle;
        }

        #endregion

        #region Method Region

        public override void Update(GameTime gameTime)
        {
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
        }

        public void DrawLevel(SpriteBatch spriteBatch, Camera camera)
        {
            levels[currentLevel].Draw(spriteBatch, camera);
        }

        #endregion
    }
}
```

I added a readonly field to the class, **levels**, is a **List<Level>**. I also added in a field, **currentLevel**, that returns the current level the player is on. There is a get only property to expose the levels, **Levels**. There is also a property to expose the **currentLevel** field, **CurrentLevel**. The get part just returns the **currentLevel** field. The set part throws an **IndexOutOfBounds** exception if you try and set **currentLevel** to an inappropriate value. It will also throw an exception if the level at the index is null. It then sets the **currentLevel** field to the value passed in. The **DrawLevel** method draws the current level. It has for parameters a **SpriteBatch** and **Camera** objects. It just calls the **Draw** method of the **Level** class.

The **GamePlayScreen** needs to be updated to use the **World** class. I also moved the code for controlling the sprite and camera into the **Player** class. I also updated the constructor to take an **AnimatedSprite** parameter for the sprite of the player. I also added the code to draw the sprite to the **Player** class. Change the **Player** class to the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

using XRpgLibrary;
using XRpgLibrary.TileEngine;
using XRpgLibrary.SpriteClasses;

namespace EyesOfTheDragon.Components
{
    public class Player
    {
        #region Field Region

        Camera camera;
        Game1 gameRef;
        readonly AnimatedSprite sprite;

        #endregion

        #region Property Region

        public Camera Camera
        {
            get { return camera; }
            set { camera = value; }
        }

        public AnimatedSprite Sprite
        {
            get { return sprite; }
        }

        #endregion

        #region Constructor Region

        public Player(Game game, AnimatedSprite sprite)
        {
            gameRef = (Game1)game;
            camera = new Camera(gameRef.ScreenRectangle);
            this.sprite = sprite;
        }

        #endregion

        #region Method Region

        public void Update(GameTime gameTime)
        {
            camera.Update(gameTime);
            sprite.Update(gameTime);

            if (InputHandler.KeyReleased(Keys.PageUp) ||
                InputHandler.ButtonReleased(Buttons.LeftShoulder, PlayerIndex.One))
            {
                camera.ZoomIn();
                if (camera.CameraMode == CameraMode.Follow)
```

```
            camera.LockToSprite(sprite);
        }
        else if (InputHandler.KeyReleased(Keys.PageDown) ||
            InputHandler.ButtonReleased(Buttons.RightShoulder, PlayerIndex.One))
        {
            camera.ZoomOut();
            if (camera.CameraMode == CameraMode.Follow)
                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 (camera.CameraMode == CameraMode.Follow)
                camera.LockToSprite(sprite);
        }
        else
        {
            sprite.IsAnimating = false;
        }

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

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

```
        }

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            sprite.Draw(gameTime, spriteBatch, camera);
        }

        #endregion
    }
}
```

You will notice that I have the **sprite** field set to readonly. That means it can't be assigned to other than as a class initializer or the constructor. You can still interact with it though, like calling a method or setting a field. I also added a property to expose the sprite outside of the **Player** class. The one thing I will mention about the **Update** method is that you should update the sprite after you update the camera. The **Draw** method just calls the **Draw** method of the **sprite** field.

I also changed the **GamePlayScreen** class to use the **World** class. Change the **GamePlayScreen** to the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

using EyesOfTheDragon.Components;

using XRpgLibrary;
using XRpgLibrary.TileEngine;
using XRpgLibrary.SpriteClasses;
using XRpgLibrary.WorldClasses;

namespace EyesOfTheDragon.GameScreens
{
    public class GamePlayScreen : BaseGameState
    {
        #region Field Region

        Engine engine = new Engine(32, 32);
        Player player;
        World world;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public GamePlayScreen(Game game, GameStateManager manager)
            : base(game, manager)
        {
            world = new World(game, GameRef.ScreenRectangle);
        }

        #endregion

        #region XNA Method Region

        public override void Initialize()
        {
```

```
            base.Initialize();
        }

        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);

            AnimatedSprite sprite = new AnimatedSprite(spriteSheet, animations);
            player = new Player(GameRef, sprite);

            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);

            TileMap map = new TileMap(tilesets, mapLayers);
            Level level = new Level(map);
            world.Levels.Add(level);
            world.CurrentLevel = 0;
        }

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

            base.Update(gameTime);
        }

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

            world.DrawLevel(GameRef.SpriteBatch, player.Camera);

            player.Draw(gameTime, GameRef.SpriteBatch);

            base.Draw(gameTime);

            GameRef.SpriteBatch.End();
        }

        #endregion

        #region Abstract Method Region
        #endregion
    }
}
```

I first remove the **sprite** field as it is no longer needed. I also added in a **World** field. You can no longer create the **Player** object in the constructor so that was moved to the **LoadContent** method. There is a local variable in the **LoadContent** method to hold the sprite for the player. After creating the sprite I create the **Player** object passing in the **GameRef** field and the sprite I just created. There is now a local variable of type **TileMap**. I then create a **Level** object using the **TileMap**. I add the level to the list of levels in the **World** object. I then set the **CurrentLevel** property to be 0, the only level there is.

The **Update** method just calls the **Update** method of the **Player** object. The **Update** method now just calls the **Update** method of the **Player** class. The **Draw** method now calls the **Draw** method of the **Player** object after calling the **DrawLevel** method of the **World** object.

The tile engine isn't all that efficient at the moment. Every tile in the map is being drawn and there are far more tiles in the map than can fit on the screen. It would be much better to only draw the visible tiles, plus and minus 1 tile. The reason I say plus and minus 1 is if the sprite is in the middle of a tile and you don't add 1 tile on the right side the blue background will show through, the same is true for the bottom. The same is true for the left and top except you subtract 1 instead of adding it. Change the **Draw** method of the **TileMap** class to the following.

```
public void Draw(SpriteBatch spriteBatch, Camera camera)
{
    Point cameraPoint = Engine.VectorToCell(camera.Position * (1 / camera.Zoom));
    Point viewPoint = Engine.VectorToCell(
        new Vector2(
            (camera.Position.X + camera.ViewportRectangle.Width) * (1 / camera.Zoom),
            (camera.Position.Y + camera.ViewportRectangle.Height) * (1 / camera.Zoom)));

    Point min = new Point();
    Point max = new Point();

    min.X = Math.Max(0, cameraPoint.X - 1);
    min.Y = Math.Max(0, cameraPoint.Y - 1);
    max.X = Math.Min(viewPoint.X + 1, mapWidth);
    max.Y = Math.Min(viewPoint.Y + 1, mapHeight);

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

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

            for (int x = min.X; x < max.X; 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);
            }
        }
    }
}
```

The fact that I allowed the camera to zoom in and out makes life a little more difficult for you. If you are zooming in, zoom > 1, you are showing less of the map. If you are zooming out, zoom < 1, you are showing more of the map. That affects where to start and stop drawing tiles. You need to use the inverse, 1 divided by a value, of the zoom value to control where to start and stop drawing tiles.

There are four **Point** variables in this class. The first, **cameraPoint**, is the tile the camera is in. The second, **viewPoint**, is the tile the camera plus the size of the view port. To find the tile the camera is in you take the camera's position and multiply it by the inverse of the zoom value of the camera. To find the tile the camera is in plus the size of the screen you do the same. For the X value you take the X value of the camera's position and add the width of the view port. You then multiply that value by the inverse of the zoom value. Similarly, for the Y value you take the Y value of the camera's position and add the height of the view port. You then multiply that value by the inverse of the zoom value.

The other two points are **min** and **max**. They hold the start and ending values of where to start and stop drawing tiles. I use the **Math.Max** method to determine where to start drawing tiles from. It returns the maximum of the two values passed in. I pass in 0 and **cameraPoint** minus 1 for both X and Y. If the camera starts out in tile (0, 0) and up just subtract 1 you will generate an **IndexOutOfBounds** exception that the game will crash. To determine the values of **max** I used the **Math.Min** method. This

method returns the minimum of the two values passed in. I pass in the **viewPoint** plus 1 and the width or height of the map. Now in the for loops that I do the drawing from I start the outer loop at **min.Y** and end at **max.Y**. Similarly, for the inner loop I start at **min.X** and end at **max.X**.

There has been a request for a list box control on my forum. I was going to add in a list box to the controls in this tutorial. There is a slight problem though. I've been using the up and down keys to move between controls on the screen. That means I can't easily use the up and down keys to scroll up and down in the list box as well. So, I'm going to hold off on that for this tutorial. I plan on adding in a list box control though.

I'm going to end this tutorial here as I don't want to get into anything new. I want to try and keep the tutorials 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