Creating a Role Playing Game with XNA Game Studio 3.0
Part 5
Scrolling the Map

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 Role Playing Game Tutorials You will also find the latest version of the project on the web site. 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: XNA RPG 4

Today I'm going to add scrolling the tile map. A few of these techniques are thanks to Nick Gravelyn. He has an excellent set of video tutorials on creating a tile engine on his web site. (To visit his site, just click his name above.)

The first thing I'm going to do is add a new class to the **CoreComponents** folder called **Camera**. I'm going to implement a simple 2D camera to help scroll the map. I will give you the code and then try and explain why I did what I did. This is the code for the camera class:

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

namespace New2DRPG.CoreComponents
{
    class Camera
    {
        public Vector2 Position;
        float speed;

        public Camera(Vector2 position, float speed)
        {
            this.Position = position;
            this.speed = speed;
        }

        public Camera()
        {
            this.Position = Vector2.Zero;
            this.speed = 3.0f;
        }

        public float Speed
        {
            get { return speed; }
            set
            {
                speed = MathHelper.Clamp(value, 0.5f, 50);
            }
        }
```

```
        public void LockCamera()
        {
            Position.X = MathHelper.Clamp(
                Position.X,
                0,
                TileEngine.MapWidthInPixels - TileEngine.ScreenWidth);
            Position.Y = MathHelper.Clamp(
                Position.Y,
                0,
                TileEngine.MapHeightInPixels - TileEngine.ScreenHeight);
        }
    }
}
```

The first thing I had to do was add a **using** statement for the XNA Framework because I needed the **Vector2** class, as well as the **MathHelper** class. There are two fields in this class, a public variable for the position of the camera and a private variable for the speed the camera moves at. I made the position public because you can't access the members of a property out side of the class. You can't do the following code out side of the class:

```
Position.Y = mapHeight() - TileEngine.ScreenHeight;
```

There are two constructors for this class. The first one accepts the position and speed of the camera. The second one sets the position of the camera in the upper left hand corner and sets the speed to 3.0f. This isn't a special number, I just like the speed at which the map scrolls with this speed.

There is one property in this class. It is a get and set for the speed of the camera. The set uses a method of the **MathHelper** class called **Clamp**. If you are not familiar with that method, it makes sure that value passed is between the minimum and maximum values passed to it.

There is one method in this class **LockCamera**. What this method does is makes sure the player can not scroll the map off the screen showing the blue back ground color. I used the **Clamp** method to make sure that the X and Y values of the camera position where in the range I wanted. The minimum value is of course zero because if the position of the camera is negative it would be outside the map.

The maximum might be a little harder to understand. To keep the map from scrolling off the right side of the map you need to subtract the width of the screen. The same is true for the bottom of the screen, except you need to subtract the height of the screen.

In the tile engine, which I will get to in a moment, I have a few static variables. These are to be helpers to other classes that need to know things about the map, like the **Camera** class. I will get to those in just a moment.

There were many changes to the **TileEngine** class. As always, I will give you the new code and try and explain why I did what I did. This is the new **TileEngine** 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.CoreComponents
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class TileEngine : Microsoft.Xna.Framework.DrawableGameComponent
    {
        SpriteBatch spriteBatch;
        Texture2D tileset;
        Rectangle[] tiles = new Rectangle[4];
        KeyboardState oldState;
        KeyboardState newState;

        int[,] map;

        static int tileWidth = 80;
        static int tileHeight = 60;

        static int tileMapWidth;
        static int tileMapHeight;

        static int screenWidth;
        static int screenHeight;

        static int mapWidthInPixels;
        static int mapHeightInPixels;

        Random random = new Random();
        Camera camera = new Camera();

        public TileEngine(Game game, Texture2D tileset)
            : base(game)
        {
            map = new int[50, 50];

            for (int x = 0; x < 50; x++)
                for (int y = 0; y < 50; y++)
                    map[y, x] = random.Next(0, 4);

            spriteBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));

            this.tileset = tileset;
```

```
        tiles[0] = new Rectangle(0, 0, 128, 128);
        tiles[1] = new Rectangle(128, 0, 128, 128);
        tiles[2] = new Rectangle(0, 128, 128, 128);
        tiles[3] = new Rectangle(128, 128, 128, 128);
        screenWidth = game.Window.ClientBounds.Width;
        screenHeight = game.Window.ClientBounds.Height;
    }

    public static int TileMapWidth
    {
        get { return tileMapWidth; }
    }

    public static int TileMapHeight
    {
        get { return tileMapHeight; }
    }

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

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

    public static int ScreenHeight
    {
        get { return screenHeight; }
    }

    public static int MapWidthInPixels
    {
        get { return mapWidthInPixels; }
    }

    public static int MapHeightInPixels
    {
        get { return mapHeightInPixels; }
    }

    public override void Initialize()
    {
        // TODO: Add your initialization code here

        base.Initialize();
    }

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

```
tileMapWidth = map.GetLength(1);
tileMapHeight = map.GetLength(0);

mapWidthInPixels = tileMapWidth * tileWidth;
mapHeightInPixels = tileMapHeight * tileHeight;

newState = Keyboard.GetState();
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();

            oldState = newState;
        }

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

            for (int x = 0; x < tileMapWidth; x++)
            {
                for (int y = 0; y < tileMapHeight; y++)
                {
                    spriteBatch.Draw(tileset,
                        new Rectangle(x * tileWidth - (int)camera.Position.X,
                            y * tileHeight - (int)camera.Position.Y,
                            tileWidth,
                            tileHeight),
                        tiles[map[y, x]],
                        Color.White);
                }
            }

        }

        public virtual void Show()
        {
            Enabled = true;
            Visible = true;
        }

        public virtual void Hide()
        {
            Enabled = false;
            Visible = false;
        }

        public virtual void Pause()
        {
            Enabled = !Enabled;
        }
    }
}
```

The first new thing you will see are two new variables to hold the current and last keyboard states. I also got rid of the huge array intialization. Instead I'm going to create the array in the contructor. Next there are eight static variables. The reason they are static is because I want to be able to use them easily outside of the class. I don't want them to be able to be modified them outside the class so I made public get only properties to use them. There are variables for the tile width and height, the height and width of the map, the size of the screen and the height and width of the map in pixels. There are also a **Camera** object and a **Random** object.

In the constructor things have changed a little. I created a fairly large map, fifty tiles by fifty tiles and set the tile to one of the four random tiles. I also use the **Game** object that was passed to the constructor to get the height and the width of the screen.

There are eight get only properties to corrosponde the variables with the same. The properties have their first letter capitalized and the variables have their first letter in lower case.

A lot of changes were add into the **Update** method. The first thing I did was get the width and the height of the map in tiles as well as the width and height of the map in pixels. I also get the state of the keyboard.

You will see a **Vector2**. This vector will be used to smooth the motion when scrolling the on diagonals. If you look at a right angled triangle, if you take the distance from moving up one unit and over one unit the distance is longer than one unit. You can normalize the vector that will make it one unit in what ever direction you are trying to move in.

Now you will see eight if statements. If the **Up** key or the **8** key on the numeric keypad the motion vector is set to -1 in the Y direction, as to move up the screen you need to subtract 1 from the current Y coordinate. This is because the pixels on your screen work like then. The X increases from left to right as you go across the screen. Y increases as you go from the top to the bottom of the screen.

For the **Right** and **6** key on the number pad, the X direction is increased by 1. **Down** and **2**, the Y direction is increased. **Left** and **4**, X is decreased by 1. The other four are for the **9**, **3**, **1** and **7** keys on the number pad. These will move the camera up and right, down and right, down and left and up and left respectively.

After finding the direction the camera will move, the vector is normalize, if it isn't the **Zero** vector. If you try and normalize the **Zero** vector it will cause an exception and crash your game. After normalizing the vector I multiply it by the speed of the camera and add it to the position of the camera. Then I lock the camera to the screen. Finally I set the old state of the keyboard to the current state of the keyboard.

The final change is in the **Draw** method. The only change is determing where to draw the tiles on the screen. You would think you would add the position of the camera to the X and Y coordinates. It doesn't work this way however. If you move the screen right, the camera actually moves to the left.

Well, that is yet another tutorial. Try and visit again soon and I will hopefully have something new on my web site. Thank you again for reading these tutorials.