Creating a Role Playing Game with XNA Game Studio 3.0
Part 15
Switching Tile Engine to use a View Port

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 14 You can download the graphics for the these tutorials at this link: Graphics.zip

Again, I did quite a bit of programming on Eyes of the Dragon, the name I have given to the game that I am creating to teach one way on how to create a role playing game with XNA 3.0. This time I have modified the tile engine to use a view port instead of drawing to the entire screen. This will be helpful later on when I create what I call a heads up display(HUD) for the game. I made a small graphic to represent the HUD of the game. I just picked a texture from Genetica Viewer 3 and cut out a 800 pixel by 600 pixel area to represent where the map will be drawn.

To get started with this tutorial the first thing you will have to do, if you are going to follow along as you read this, is load up the last version of Eyes of the Dragon. You will also need the new graphic that I created. You can find it in this file: Graphics.zip

Open the **Content** folder then right-click the **Backgrounds** folder and select **Add existing item**. Find the **characterhud.png** file, from the graphics file, and add it to the folder. Now you will need to open the **ActionScreen** class. There were only a few minor changes to the class but I will give you the new code and then explain why I did what I did. This is the new **ActionScreen** class.

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

namespace New2DRPG
{
    class ActionScreen : GameScreen
    {
        SpriteFont gameFont;
        Texture2D chest;
        Texture2D tilesetTexture;
        Texture2D characterHUDTexture;

        ContentManager Content;
        SpriteBatch spriteBatch;

        string tilesetName;
        TileEngine tileEngine;
        Tileset tileset;
```

```
        public ActionScreen(Game game, SpriteFont gameFont, string tilesetName)
            : base(game)
        {
            this.gameFont = gameFont;
            Content =
                (ContentManager)Game.Services.GetService(typeof(ContentManager));

            spriteBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
            this.tilesetName = tilesetName;
            LoadContent();

            tileset = new Tileset(tilesetTexture, 128, 128, 4, 4);
            tileEngine = new TileEngine(game, this.tileset, 50, 50);
            Components.Add(tileEngine);
            tileEngine.Show();

        }

        protected override void LoadContent()
        {
            base.LoadContent();
            tilesetTexture = Content.Load<Texture2D>(@"Tilesets\" + tilesetName);
            chest = Content.Load<Texture2D>(@"Items\chest");
            characterHUDTexture =
                  Content.Load<Texture2D>(@"Backgrounds\characterhud");
        }

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

        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
            Vector2 position = Vector2.Zero;
            spriteBatch.Draw(characterHUDTexture, position, Color.White);
        }

        public override void Show()
        {
            base.Show();
            Enabled = true;
            Visible = true;
        }

        public override void Hide()
        {
            base.Hide();
            Enabled = false;
            Visible = false;
        }
    }
}
```

As you can see, I added in a new variable to hold the texture of the HUD. Then, in the **LoadContent**

method I loaded in the texture for the HUD. The last change is in the **Draw** method. In this method I will actually draw the HUD. Now, I draw the HUD after calling **base.Draw(gameTime)** so that the HUD will be drawn after everything else. This isn't a big issue because the area of the HUD where the map will be drawn is transparent.

To draw to just the view port, instead of the whole screen, I had to make some changes to the **TileEngine**, **TileMap** and **Camera** classes as well. First I will deal with the **TileEngine** class. The **TileEngine** class is getting long but I will give you the entire class and then explain the changes that I had to make to the **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;
using New2DRPG.SpriteClasses;

namespace New2DRPG.CoreComponents
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class TileEngine : Microsoft.Xna.Framework.DrawableGameComponent
    {
        SpriteBatch spriteBatch;
        ContentManager Content;
        static List<Rectangle> tiles = new List<Rectangle>();
        KeyboardState oldState;
        KeyboardState newState;

        TileMap map;

        List<GameComponent> childComponents = new List<GameComponent>();

        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;

        static float cameraX;
        static float cameraY;

        static int viewPortWidth = 800;
```

```
static int viewPortHeight = 600;

Random random = new Random();
Camera camera = new Camera();
static Tileset tileset;

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

    TileEngine.tileset = tileset;
    TileEngine.tileMapWidth = tileMapWidth;
    TileEngine.tileMapHeight = tileMapHeight;

    TileEngine.tiles = tileset.Tiles;

    map = new TileMap(tileMapWidth, tileMapHeight, game);
    childComponents.Add(map);

    Content =
        (ContentManager)Game.Services.GetService(typeof(ContentManager));

    Texture2D chest = Content.Load<Texture2D>(@"Items\chest");

    ItemSprite tempItemSprite;
    Vector2 position;
    Random random = new Random();

    for (int i = 0; i < 3; i++)
    {
        position = new Vector2(
            random.Next(0, 10),
            random.Next(0, 10));

        tempItemSprite = new ItemSprite(game,
            chest, position);

        childComponents.Add(tempItemSprite);
    }

    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 static int CameraXPosition
{
    get { return (int)cameraX; }
}

public static int CameraYPosition
{
    get { return (int)cameraY; }
}

public static Vector2 CameraVector
{
    get { return new Vector2(cameraX, cameraY); }
}

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; }
```

```
}

public static Texture2D TileSet
{
    get { return Tileset.TilesetTexture; }
}

public static List<Rectangle> Tiles
{
    get { return tiles; }
}

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

    base.Initialize();
}

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

    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();
        TileEngine.cameraX = camera.Position.X;
        TileEngine.cameraY = camera.Position.Y;
        oldState = newState;
    }

    public override void Draw(GameTime gameTime)
    {
        foreach (GameComponent child in childComponents)
        {
            if ((child is DrawableGameComponent) &&
                ((DrawableGameComponent)child).Visible)
            {
                ((DrawableGameComponent)child).Draw(gameTime);
            }
        }

        base.Draw(gameTime);
    }

    public virtual void Show()
    {
        Enabled = true;
        Visible = true;
        foreach (GameComponent child in childComponents)
        {
            child.Enabled = true;
            if ((child is DrawableGameComponent))
                ((DrawableGameComponent)child).Visible = true;
        }
    }

    public virtual void Hide()
    {
        Enabled = false;
        Visible = false;
        foreach (GameComponent child in childComponents)
```

```
            {
                child.Enabled = false;
                if ((child is DrawableGameComponent))
                    ((DrawableGameComponent)child).Visible = false;
            }
        }

        public virtual void Pause()
        {
            Enabled = !Enabled;
            foreach (GameComponent child in childComponents)
            {
                child.Enabled = !child.Enabled;
            }
        }
    }
}
```

The first change I made was to change the size of the tiles. This step wasn't necessary but I wanted the tiles to fit nicely into the view port. Since the view port that I created was 800 pixels by 600 pixels and I wanted to keep the 4:3 pixel ratio I made the tiles 80 pixels wide by 60 pixels high. The next thing I had to do was create to static variables to hold the width and height of the view port. These were 800 pixels wide by 600 pixels high.

I also added in four public get only properties. Two of them were **Vector2**'s. One of them will return, as a **Vector2**, the position of the camera. The reason of this will become clear when I get to the **TileMap** class. I will use it to only draw the portion of the map that fits into the view port instead of drawing the entire map. This will speed up the rendering process quite a bit. The other **Vector2** property returns the size of the view port plus 1 extra tile wide and 1 extra tile high. The reason for this is because when you scroll the map if you don't pad the edges the background color will show. Again, this will be used in the **TileMap** class in determining what portion of the map to draw. The other two properties will be used in the **Camera** class. Instead of locking the camera to screen I will need to lock the camera to the view port so I needed properties to expose the height and width of the view port.

Now I will give you the code for the **Camera** class and explain the changes. This is the new **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.ViewPortWidth);
            Position.Y = MathHelper.Clamp(
                Position.Y,
                0,
                TileEngine.MapHeightInPixels - TileEngine.ViewPortHeight);
        }
    }
}
```

The only change here is in the **LockCamera** method. In this method instead of using screen width and height to lock the camera I use the two new view port properties instead. This will keep the map from scrolling off the edge of the view port.

The last class that I had to change was the **TileMap** class as this is the class where I actually do the rendering of the map. This is the new **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.CoreComponents
{
    /// <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;

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

            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(tileMapWidth, tileMapHeight);

            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, TileEngine.Tiles.Count - 1);
                    }
                }

            tileMapLayers.Add(layer);
        }

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

            base.Initialize();
        }

        public override void Update(GameTime gameTime)
        {
            // TODO: Add your update code here

            base.Update(gameTime);
        }

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

        public override void Draw(GameTime gameTime)
        {
            Point cameraPoint = VectorToCell(TileEngine.CameraVector);
            Point viewPoint = VectorToCell(TileEngine.CameraVector +
                                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, TileEngine.TileMapWidth);
            max.Y = (int)Math.Min(viewPoint.Y, TileEngine.TileMapHeight);

            foreach (TileMapLayer layer in tileMapLayers)
            {
                for (int x = min.X; x < max.X; x++)
                {
                    for (int y = min.Y; y < max.Y; y++)
                    {
                        if (layer.GetTile(x, y) != -1)
                        {
                            spriteBatch.Draw(TileEngine.TileSet,
                                new Rectangle(x * TileEngine.TileWidth -
```

```
                                        TileEngine.CameraXPosition,
                                    y * TileEngine.TileHeight -
                                        TileEngine.CameraYPosition,
                                    TileEngine.TileWidth,
                                    TileEngine.TileHeight),
                                TileEngine.Tiles[layer.GetTile(x, y)],
                                Color.White);
                        }
                    }
                }
            }
            base.Draw(gameTime);
        }

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

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

The first change I made was that in the constructor instead of choosing a random tile for the map is that I selected the first tile, the grass tile to try and give the map a better look. Next when I created the rock layer I made the layer have more rocks by instead of just having a 1 in 50 chance a 1 in 10 chance by checking to see it the random number generated was less than 5.

I added in a new method to the class. In this method I take a **Vector2** and return a **Point** that describes what tile the **Vector2** is in. I will use this method to determine which portion of the map to draw. In the **Draw** method. To find out what tile a **Vector2** is in, you take the X value of the vector and divide it by the width of the tile to get the tile's X value. You do something similar with the Y value of the vector. Instead of using the width of the tile you use the height of the tile.

The **Draw** method is going to take a bit of explaining. If you remember, the **Camera** object is used to control scrolling the map. When I lock the camera it will not scroll past the left edge of the map or the top edge of the map by making sure it is never X and Y values are never less than zero. The other edges are a little more complicated to deal with. To keep the map from scrolling of the right edge of the map I used to check to make sure the X position of the camera did not go beyond the width of the map in pixels minus the width of the screen. I did a similar thing with the bottom of the map. I checked to make sure that the Y position of the camera did not go beyond the height of the map in pixels minus the height of the screen. Now, instead of the screen I used the view port.

To determine which tiles are visible you take the position of the camera and the position of the camera plus the height and width of the map plus 1 extra tile. Now, with the method I created, I can find out what tile the camera is in by taking it's X value and dividing it by the tile width and it's Y value and dividing it by the tile height. You know that the camera will never go into negatives so you can find out what tile to start drawing with by finding out which cell the camera is in.

Just like figuring out where to stop the camera from going off the right and bottom edges is harder than stopping the camera from going of the left and top edges, finding when to stop drawing tiles is also harder. This is what you can do. You can take the position of the camera plus the size of the view port plus 1 tile on either side and find the minimum of that cell with the width and height of the map in tiles. That way you will never try and draw a tile outside of the map.

Now that you know which tile to start drawing in and which tile to stop drawing at you can use the minimum X tile and loop through tiles until you get to the maximum X tile. The same is true for the Y tile. You start at the minimum Y tile and loop through the tiles until you get to the maximum Y tile.

That is what I did to lock the map to a view port instead of the entire screen and just draw what is visible. Well, that is it for another tutorial. I will be working on more so I encourage you to keep either visiting this site or my blog, http://xna-rpg.blogspot.com for the latest news on these tutorials.