Creating a Role Playing Game with XNA Game Studio 3.0
Part 4
Adding the Action Screen
and
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 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 3

In this tutorial I will be adding a new screen to the game, the action screen, and a new **Game Component**, a simple tile engine with a few textures. The action screen, as the name implies, is where the action of the game will take place. This is the screen where the player will interact with the game world. The tile engine is what will draw the maps.

I've tried to make the tile engine so it will be easy to exctract and add to another project. (I actually took it out of a game I'm working on called Crystal Caverns. You can read about it on my web site where this tutorial is hosted.) To get started you will want to add a new **GameComponent** to the **CoreComponents** folder and call it **TileEngine**. As always I will give you the code and then explain it.

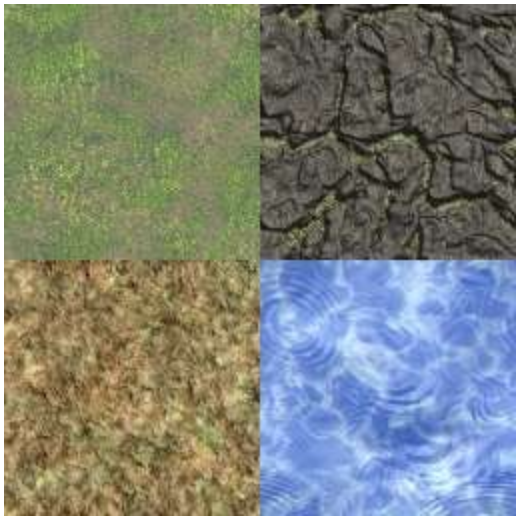
```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using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


namespace New2DRPG.CoreComponets
{
    /// <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];

        int[,] map = new int[,] {
            { 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
            { 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
```

```
    { 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, },
};

int tilewidth = 80;
int tileheight = 60;

public TileEngine(Game game, Texture2D tileset)
    : base(game)
{
    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);
}

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

    base.Initialize();
}

public override void Update(GameTime gameTime)
{

    base.Update(gameTime);
}

public override void Draw(GameTime gameTime)
{
    base.Draw(gameTime);
    int mapWidth = map.GetLength(1);
    int mapHeight = map.GetLength(0);

    for (int x = 0; x < mapWidth; x++)
    {
        for (int y = 0; y < mapHeight; y++)
        {
            spriteBatch.Draw(tileset,
                new Rectangle(x * tilewidth,
                    y * tileheight,
```

```
                        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;
    }
  }
}
```

This is a visual **Game Component** so it had to be inherited from **DrawableGameComponent** instead of **GameComponent**. There are a few variables in the class. Since drawing has to be done, you will need a **SpriteBatch** object.

There is a variable to hold a tile set. Instead of loading individual tiles, I decided to use a tile set. The tile set I made has four 128 by 128 pixel tiles, in a 256 by 256 square. The tiles were made with the help of Genetica Viewer by Spiral Graphics. It is an amazing tool for creating tile textures. It is free to use if you add a link to their web site in your game or documentation, like I just did. You can purchase a license and remove this restriction.



Sample Tile Set

Open Source Role Playing Game Home Page

There is an array of **Rectangles**. This array will hold the position of the tiles in the tile set. Right now I code them manually. Eventually, when I make the map editor, you will be able to define the rectangles for the tile set.

Next there is a 2 dimensional array of integers. This is array is used ot test the tile engine to see if it does what it should. I just filled the array to 0, which will be the first tile in the map and add three smal squares to show all of the other tiles. (I will explain the tiling process when I get to the **Draw** method.)

Finally there are two integers. One for the width of the tile on the screen, not in the tile set, and the height of the tile. Screens are wider than they are high, usually in a 4:3 ratio. I used this ration to set the size of the tiles on the screen.

The constructor for the tile engine takes two parameters, a **Game** object and a **Texture2D** object that is the tile set. I get the **SpriteBatch** object for the component the same way I did in the other components, using **Game Services**. Next, I set the tile set. Then, I set the **Rectangles** for the tiles.

Right now there are no changes in the **Initialize** and **Update** methods. There is a change in the **Draw** method though. This is where I will draw the map.

The first thing you need to do is get the width and height of the map. This may seem a little backwards to you. When you are tiling a map, the width of the map is the second dimension of the array and the height of the map is the first dimension of the array.

To draw the map you loop through the map using nested loops. I used x for the width of the map and y for the height of the map. You find the x coordinate of the tile by taking the x index and multiplying it by the width of the tile. To find the y coordinate you do something similar. You take the y index and multiply it by the height of the tile.

To draw the tile, you call the **Draw** method of the **SpriteBatch** object. The overload I used takes four parameters. The texture to be drawn, where the tile is to be drawn as a **Rectangle**, the source **Rectangle** in the texture to be drawn and the tint color.

There are three virtual methods in this component. One to show the screen, one to hide the screen and one to pause it. When you pause the screen it will still be visible but it won't update itself.

Now it is time to your attention to the action screen. I've created the action screen so you can have multiple action screens. All that you will need to do is just create a new action screen for different maps.

You will want to add a new class to the project called **ActionScreen**. The action screen will have a **TileEngine** game component. This component will draw the map. The code for the action screen is like the other screens. I will show you the code then explain it.

```
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;

namespace New2DRPG
{
    class ActionScreen : GameScreen
    {
        SpriteFont gameFont;
        Texture2D tileset;
        ContentManager Content;
        SpriteBatch spriteBatch;

        string tilesetName;
        TileEngine tileEngine;

        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();

            tileEngine = new TileEngine(game, this.tileset);
            Components.Add(tileEngine);
            tileEngine.Show();
        }

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

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

        }

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

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

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

You will see that there is a new **using** statement in this class. I decide that instead of loadin all of the content in the **Game1** class, to load the tile sets when the action screen is created. You need a **ContentManager** object to load in your content. Just like you did with the **SpriteBatch** object, you can add a **ContentManager** to the **Game Services**. I also inherited this class from **GameScreen**.

There are several variables in this class. There is a variable to hold a **SpriteFont** and a **SpriteBatch** object for drawing sprites and text. They are not used right now but they may be needed later so I added them in. I created a **Texture2D** to hold the tile set to be loaded. Like I said above, you need a **ContentManager** to load the tile set. There is a **string** to hold the name of the tile set and a **TileEngine** game component.

The constructor for this class has three parameters. A **Game** object, **SpriteFont** and a **string** to hold the name of the tile set to be loaded. In the constructor, I get the **ContentManager** from the list of **Game Services** the same way I get the **SpriteBatch** object. After getting the **ContentManager** and **SpriteBatch** objects and setting the name of the tile set, I call the **LoadContent** method. After that, I create a **TileEngine** object, add it to the list of components and show it.

The **LoadContent** method just calls the **LoadContent** method of the base class and loads in the tile set. You will see that the tile sets are in a sub folder in the **Content** folder. I will get to that in just a moment when I get to the **Game1** class.

The **Update** and **Draw** methods just call the base class. The **Show** and **Hide** methods call the **Show** and **Hide** methods of the base class and show and hide the action screen.

There are a few things that you need to do in the **Game1** class. First, you need to add a new folder to the **Content** folder called **Tilesets**. You will need to add the tile set to the project. You can find it HERE. I will go over the changes to the **Game1** class and give you the code afterwards.

First, I created a variable to hold the action screen. In the **LoadContent** method I added the **ContentManager** object to the list of services. Also in the **LoadContent** method I create the action screen, add it to the list of components and hide it. The last two changes are in the **HandleStartScreenInput** method and the **HandleCreatePCScreenInput** methods. All I did in the **HandleStartScreenInput** method was add a new case for the "**The Story Continues**" menu choice. I just hide the active screen, set the active screen to the action screen and show the action screen. I did the same thing for the **HandleCreatePCScreenInput** method for the **"Begin the Adventure**" menu option.

The new version of the **Game1** class begins on the next page. Thank you for reading this tutorial, I will try and have another one ready shortly.

```
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.CoreComponents;

namespace New2DRPG
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        StartScreen startScreen;
        CreatePCScreen createPCScreen;
        HelpScreen helpScreen;
        ActionScreen actionScreen;

        GameScreen activeScreen;

        SpriteFont normalFont;

        Texture2D background;

        KeyboardState newState;
        KeyboardState oldState;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before
starting to run.
        /// This is where it can query for any required services and load any non-
graphic
        /// related content.  Calling base.Initialize will enumerate through any
components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            base.Initialize();
        }
```

```
        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
            Services.AddService(typeof(SpriteBatch), spriteBatch);
            Services.AddService(typeof(ContentManager), Content);

            normalFont = Content.Load<SpriteFont>("normal");
            createPCScreen = new CreatePCScreen(this, normalFont);
            Components.Add(createPCScreen);

            background = Content.Load<Texture2D>("gryphon");
            startScreen = new StartScreen(this, normalFont, background);
            Components.Add(startScreen);

            background = Content.Load<Texture2D>("fire-dragon");
            helpScreen = new HelpScreen(this, background);
            Components.Add(helpScreen);

            actionScreen = new ActionScreen(this, normalFont, "tileset1");
            Components.Add(actionScreen);
            actionScreen.Hide();

            startScreen.Show();
            helpScreen.Hide();
            createPCScreen.Hide();

            activeScreen = startScreen;
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// all content.
        /// </summary>
        protected override void UnloadContent()
        {
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Update(GameTime gameTime)
        {
            newState = Keyboard.GetState();

            if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed)
                this.Exit();

            if (activeScreen == startScreen)
            {
```

```
                HandleStartScreenInput();
            }
            else if (activeScreen == helpScreen)
            {
                HandleHelpScreenInput();
            }
            else if (activeScreen == createPCScreen)
            {
                HandleCreatePCScreenInput();
            }

            oldState = newState;

            base.Update(gameTime);
        }

        private void HandleHelpScreenInput()
        {
              if (CheckKey(Keys.Space) || CheckKey(Keys.Enter) ||
CheckKey(Keys.Escape))
            {
                activeScreen.Hide();
                activeScreen = startScreen;
                activeScreen.Show();
            }
        }

        private void HandleStartScreenInput()
        {
            if (CheckKey(Keys.Enter) || CheckKey(Keys.Space))
            {
                switch (startScreen.SelectedIndex)
                {
                    case 0:
                        activeScreen.Hide();
                        activeScreen = createPCScreen;
                        activeScreen.Show();
                        break;
                    case 1:
                        activeScreen.Hide();
                        activeScreen = actionScreen;
                        actionScreen.Show();
                        break;
                    case 2:
                        activeScreen.Hide();
                        activeScreen = helpScreen;
                        activeScreen.Show();
                        break;
                    case 3:
                        Exit();
                        break;
                }
            }
        }

        private void HandleCreatePCScreenInput()
        {
```

```
            if (CheckKey(Keys.Enter) || CheckKey(Keys.Space))
            {
                switch (createPCScreen.SelectedIndex)
                {
                    case 0:
                        createPCScreen.ChangeName();
                        break;
                    case 1:
                        createPCScreen.ChangeGender();
                        break;
                    case 2:
                        createPCScreen.ChangeClass();
                        break;
                    case 3:
                        createPCScreen.ChangeDifficulty();
                        break;
                    case 4:
                        activeScreen.Hide();
                        activeScreen = startScreen;
                        activeScreen.Show();
                        break;
                    case 5:
                        activeScreen.Hide();
                        activeScreen = actionScreen;
                        activeScreen.Show();
                        break;
                }
            }
        }

        private bool CheckKey(Keys theKey)
        {
            return oldState.IsKeyDown(theKey) && newState.IsKeyUp(theKey);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
            base.Draw(gameTime);
            spriteBatch.End();
        }
    }
}
```