Creating a Role Playing Game with XNA Game Studio 3.0
Part 38
Refactoring - Part 1

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: http://www.jtmbooks.com/rpgtutorials/New2DRPG37.zip You can download the graphics from this link: Graphics.zip

In this tutorial I am going to do a little bit of refactoring. There has been something that has been bothering me for a while now and I'm going to start the process of fixing it. I don't like the fact that I'm passing the player character and the player's sprite around. I want to try and keep them separate from the rest of the game. Something else that I will be introducing soon is the idea of a session for the game. Instead of loading maps in the **ActionScreen** class for example I am going to create a class for the session of the game. It will hold the maps, quests, items, NPCs, etc. I'm planning on eventually removing the **ActionScreen** class entirely from the game. It was a good idea for a fairly simple game but this game is rather complex and as it turns out it is probably not the best idea.

To begin make sure that the game is the current start up project. Right click your game in the solution explorer and select the **Set As Start Up Project** if it is not already. The first thing that I'm going to do is add a new **DrawableGameComponent** to the **CoreComponents** folder. Right click the **CoreComponents** folder and select **Add** and then **New Item**. Make sure that the **XNA** node is selected and choose **Game Component** and name it **PlayerComponent**. This is the code for the **PlayerComponent** 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
{
     class PlayerComponent : Microsoft.Xna.Framework.DrawableGameComponent
    {
        AnimatedSprite sprite;
        PlayerCharacter playerCharacter;
        Game game;

        public PlayerComponent(Game game, AnimatedSprite sprite, PlayerCharacter
playerCharacter)
            : base(game)
        {
            this.sprite = sprite;
            this.playerCharacter = playerCharacter;
            this.game = game;
        }

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

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

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

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

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

    }
}
```

The first thing that you will see is that I added a using statement for the **New2DRPG.SpriteClasses** namespace. This is because this component will be responsible for drawing and updating the sprite. Eventually it will also have a session class and will be responsible for handling just about everything to do with the game. Since this component will be drawing I inherited it from **DrawableGameComponent**.

There are three fields in this class: a **Game** object **game**, an **AnimatedSprite** object **sprite** and a **PlayerCharacter** object **playerCharacter**. The constructor for this class takes three parameters: a **Game** object, an **AnimatedSprite** and a **PlayerCharacter** object. The constructor then sets the fields of the class. I didn't add anything to the **Intialize** method.

In the **Update** method is where I the **PlayerCharacter** and **AnimatedSprite** are updated. I just call their **Update** methods passing in **gameTime**. I added in a **Draw** method to actually draw the player character and the sprite, again passing in **gameTime**. There are **Show** and **Hide** methods to show and hide the component.

I made three small changes to the **PlayerCharacter** class. The first change is that I made it an abstract class. That means that you can not create an instance of the **PlayerCharacter** class. You can assign to it classes that were created by classes that inherit form it though. I only ever want classes that inherit from **PlayerCharacter** to be created. Since it will no longer be responsible for drawing the sprite so I removed the calls to draw and update the sprite from the **Update** and **Draw** methods. These are the **Draw** and **Update** methods for the **PlayerCharacter** class as well as the new class declaration.

```
abstract class PlayerCharacter : Microsoft.Xna.Framework.DrawableGameComponent

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

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

The next thing I want to do is make a quite change to the **Sprite** class. What I want to do is add in a new property to the sprite class that will return a **Rectangle** that represents the bounds of the sprite on the screen. All that the property does is cast the **X** and **Y** values of **position** to integers and use the width and height of the sprite for the other parameters. Add the following property to the **Sprite** class.

```
public Rectangle Bounds
{
    get
    {
        return new Rectangle(
            (int)position.X,
            (int)position.Y,
            width,
            height);
    }
}
```

There were quite a few changes to the **ActionScreen** class as I started to prepare for removing it from the game. I will give you the new code for the entire class and then go over the differences. This is the code for the **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;
        SpriteFont interfaceFont;
        Texture2D chest;
        Texture2D characterHUDTexture;
        string tilesetName;
        Tileset tileset;
        int viewportWidth;
        int viewportHeight;
        int screenWidth;
        int screenHeight;

        PlayerComponent player;

        TileMap tileMap;

        List<AnimatedSprite> animatedSprites = new List<AnimatedSprite>();
        string[] assetNames =
            {
                @"Sprites\knt1",
                @"Sprites\nja1",
                @"Sprites\skl1" };
        Texture2D[] spriteTextures;
        Random random = new Random();

        public ActionScreen(Game game, SpriteFont gameFont, string tilesetName)
            : base(game)
        {
            player = new PlayerComponent(game, null, null);
            this.gameFont = gameFont;

            this.tilesetName = tilesetName;
            LoadContent();
            tileMap = new TileMap(@"Content\TileSets\tilemap.tmap", tileset, game);
            Components.Add(tileMap);
            tileMap.Show();

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

        protected override void LoadContent()
        {
            base.LoadContent();
            tileset = Content.Load<Tileset>(@"TileSets\" + tilesetName);
            chest = Content.Load<Texture2D>(@"Items\chest");
            characterHUDTexture =
Content.Load<Texture2D>(@"Backgrounds\characterhud");
            this.interfaceFont = Content.Load<SpriteFont>("smallFont");

            spriteTextures = new Texture2D[assetNames.Length];
            for (int i = 0; i < assetNames.Length; i++)
                spriteTextures[i] = Content.Load<Texture2D>(assetNames[i]);
        }

        private void CreateSprites(Game game)
        {
            List<Animation> listAnimation = new List<Animation>();

            Animation tempAnimation = new Animation(2, 64, 64, 0, 0);
            listAnimation.Add(tempAnimation);

            tempAnimation = new Animation(2, 64, 64, 128, 0);
            listAnimation.Add(tempAnimation);

            tempAnimation = new Animation(2, 64, 64, 256, 0);
            listAnimation.Add(tempAnimation);

            tempAnimation = new Animation(2, 64, 64, 384, 0);
            listAnimation.Add(tempAnimation);

            for (int i = 0; i < assetNames.Length; i++)
            {
                List<Animation> animations = new List<Animation>();

                foreach (Animation a in listAnimation)
                {
                    Animation clonedAnimation = (Animation)a.Clone();
                    animations.Add(clonedAnimation);
                }

                animatedSprites.Add(
                    new AnimatedSprite(game,
                        spriteTextures[i],
                        animations));

                animatedSprites[i].IsAnimating = true;
                int direction = random.Next(0, 4);

                switch (direction)
                {
                    case 0:
                        animatedSprites[i].CurrentAnimation = AnimationKey.Up;
                        break;
                    case 1:
                        animatedSprites[i].CurrentAnimation = AnimationKey.Down;
                        break;
                    case 2:
                        animatedSprites[i].CurrentAnimation = AnimationKey.Left;
                        break;
                    case 3:
                        animatedSprites[i].CurrentAnimation = AnimationKey.Right;
                        break;
                }

                Vector2 position = new Vector2();

                position.X = TileEngine.TileWidth * random.Next(1, 10);
                position.Y = TileEngine.TileHeight * random.Next(1, 10);

                animatedSprites[i].Position = position;
            }
```

```
}

public bool CheckUnWalkableTile(Rectangle bounds, Vector2 motion)
{
    Rectangle nextRectangle = new Rectangle(
        bounds.X + (int)motion.X,
        bounds.Y + (int)motion.Y,
        bounds.Width,
        bounds.Height);

    if (motion.Y < 0 && motion.X < 0)
    {
        return tileMap.CheckUpAndLeft(nextRectangle);
    }
    else if (motion.Y < 0 && motion.X == 0)
    {
        return tileMap.CheckUp(nextRectangle);
    }
    else if (motion.Y < 0 && motion.X > 0)
    {
        return tileMap.CheckUpAndRight(nextRectangle);
    }
    else if (motion.Y == 0 && motion.X < 0)
    {
        return tileMap.CheckLeft(nextRectangle);
    }
    else if (motion.Y == 0 && motion.X > 0)
    {
        return tileMap.CheckRight(nextRectangle);
    }
    else if (motion.Y > 0 && motion.X < 0)
    {
        return tileMap.CheckDownAndLeft(nextRectangle);
    }
    else if (motion.Y > 0 && motion.X == 0)
    {
        return tileMap.CheckDown(nextRectangle);
    }
    return tileMap.CheckDownAndRight(nextRectangle);
}

public override void Update(GameTime gameTime)
{
    base.Update(gameTime);
    player.Update(gameTime);
    foreach (AnimatedSprite sprite in animatedSprites)
        sprite.Update(gameTime);
}

public override void Draw(GameTime gameTime)
{
    base.Draw(gameTime);
    if (Visible)
    {
        Vector2 position = new Vector2(0, 0);
        position.Y = viewportHeight;
        spriteBatch.Draw(characterHUDTexture, position, Color.White);
        player.Draw(gameTime);
        foreach (AnimatedSprite sprite in animatedSprites)
```

```
                    sprite.Draw(gameTime);
                }
            }

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

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

            public void SetPlayer(PlayerComponent player)
            {
                this.player = player;
            }

        }
    }
```

The first thing I did was change the **PlayerCharacter** field to a **PlayerComponent** field called **player**. The next change was in the constructor of the **ActionScreen** class. Instead of creating a new **PlayerCharacter** object I create an instance of **PlayerComponent**.

There is a big difference in the **CheckUnWalkableTiles** method. It now takes as a parameter a **Rectangle** called **bounds** instead of an **AnimatedSprite**. It then creates the **nextRectangle** using **bounds** and **motion**. The rest of the method is the same as before.

In the **Update** method I call **player.Update** rather than **playerCharacter.Update**. The same is true for the **Draw, Show** and **Hide** methods. They call the appropriate methods of **player** rather than **playerCharacter**. Another change was I replaced the **SetPlayerCharacter** method with a **SetPlayer** method that takes a **PlayerComponent** as a parameter and set set the **player** field.

The rest of the changes were in the **Game1** class. The first change was I added a field for a **PlayerComponent**. I also changed the **AnimatedSprite** field so that it is no longer static. I also removed the property to get the **playerSprite** field. Change the **AnimatedSprite** field to the following and add this **PlayerComponent** field to the class.

```
PlayerComponent player;
AnimatedSprite playerSprite;
```

The other two changes were in the **CreatePlayerCharacter** method and the **HandleStartScreenInput** methods. The change was that both of the methods create a **PlayerComponent** and then pass it to the **ActionScreen** class. This is the code.

```
private void CreatePlayerCharacter()
{
    if (createPCScreen.CharacterClass == CharClass.Fighter)
    {
        playerCharacter = new FighterCharacter(
            createPCScreen.CharacterName,
            createPCScreen.CharacterGender,
            createPCScreen.DifficultyLevel, this);
    }
    if (createPCScreen.CharacterClass == CharClass.Priest)
    {
        playerCharacter = new PriestCharacter(
            createPCScreen.CharacterName,
            createPCScreen.CharacterGender,
            createPCScreen.DifficultyLevel, this);
    }
    if (createPCScreen.CharacterClass == CharClass.Thief)
    {
        playerCharacter = new ThiefCharacter(
            createPCScreen.CharacterName,
            createPCScreen.CharacterGender,
            createPCScreen.DifficultyLevel, this);
    }
    if (createPCScreen.CharacterClass == CharClass.Wizard)
    {
        playerCharacter = new WizardCharacter(
            createPCScreen.CharacterName,
            createPCScreen.CharacterGender,
            createPCScreen.DifficultyLevel, this);
    }
    player = new PlayerComponent(this, playerSprite, playerCharacter);
    actionScreen.SetPlayer(player);
}

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();
                playerCharacter = new FighterCharacter(
                    "Evander",
                    false,
                    Level.Normal,
                    this);
                player = new PlayerComponent(this, playerSprite, playerCharacter);
                actionScreen.SetPlayer(player);
                activeScreen = actionScreen;
                actionScreen.Show();
                break;
            case 2:
                activeScreen.Hide();
                activeScreen = helpScreen;
                activeScreen.Show();
                break;
            case 3:
                activeScreen.Hide();
                activeScreen = creditScreen;
                activeScreen.Show();
                break;
            case 4:
                activeScreen.Enabled = false;
                activeScreen = quitPopUpScreen;
                activeScreen.Show();
                break;
        }
    }
}
```

That is it for this tutorial. In the next tutorial I will finish the refactoring that I started in the tutorial. 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 my tutorials.