Creating a Role Playing Game with XNA Game Studio
Part 52
Implementing Manager Classes - 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 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://xnagpa.net/rpgtutorials/New2DRPG51.zip. You can download the graphics from this link: Graphics.zip

In this tutorial I will get started on adding in manager classes for dealing with game objects rather than keeping track of them all in the main game class. The one thing that has been bugging me for a while now is that all sprites in the game are XNA game components. This probably wasn't the best idea in the world. As more and more objects are added to the game you may start to see performance hits.

To get started, you will want to open the last version of the game. Make sure that your game project is the start up project. If it isn't right click your game and select the **Set As Start Up** option. The **Sprite** class is the base class of all other sprites in the game. The sprite class originally inherited from the **GameComponent** class. I changed it so that it no longer inherits from any class. I made the **Update** and **Draw** methods abstract methods. That means that any class that inherits from the **Sprite** class must implement **Update** and **Draw** methods with the same signature. They both still take **GameTime** parameters like before. This is the new code for the **Sprite** 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
{
    public abstract class Sprite
    {
        protected Texture2D texture;

        protected Rectangle sourceRectangle;

        protected Vector2 position;
        protected Vector2 velocity;
        protected Vector2 center;
```

```
protected float scale;
protected float rotation;

protected SpriteBatch spriteBatch;

protected int width;
protected int height;

public Sprite(Game game, Texture2D texture)
{
    spriteBatch =
        (SpriteBatch)game.Services.GetService(typeof(SpriteBatch));

    this.texture = texture;
    width = texture.Width;
    height = texture.Height;
    position = Vector2.Zero;
    velocity = Vector2.Zero;
    center = new Vector2(texture.Width / 2,
        texture.Height / 2);

    scale = 1.0f;
    rotation = 0.0f;

    sourceRectangle = new Rectangle(0,
                              0,
                              texture.Width,
                              texture.Height);
}

public Sprite(Game game, Texture2D texture, Rectangle sourceRectangle)
{
    spriteBatch =
        (SpriteBatch)game.Services.GetService(typeof(SpriteBatch));

    this.texture = texture;
    this.sourceRectangle = sourceRectangle;

    position = Vector2.Zero;
    velocity = Vector2.Zero;
    center = new Vector2(sourceRectangle.Width / 2,
        sourceRectangle.Height / 2);
    width = sourceRectangle.Width;
    height = sourceRectangle.Height;
    scale = 1.0f;
    rotation = 0.0f;
}

public abstract void Update(GameTime gameTime);
public abstract void Draw(GameTime gameTime);

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

```
                    height);
            }
        }

        public virtual Texture2D Texture
        {
            get { return texture; }
        }

        public virtual Vector2 Position
        {
            get { return position; }
            set { position = value; }
        }

        public virtual Vector2 Velocity
        {
            get { return velocity; }
        }

        public virtual Vector2 Center
        {
            get { return center; }
        }

        public Vector2 Origin
        {
            get { return position + center; }
        }

        public virtual float Scale
        {
            get { return scale; }
        }

        public virtual float Rotation
        {
            get { return rotation; }
        }

        public int Width
        {
            get { return width; }
        }

        public int Height
        {
            get { return height; }
        }
    }
}
```

The **AnimatedSprite** class inherited from the **Sprite** class so it has to provide **Update** and **Draw** methods. Since the **Sprite** class was a **DrawableGameComponent** before there were calls to **base.Update** and **base.Draw**. They just needed to be removed from those classes and were the only changes to that class. This was a relatively short class so I will give you the code for the entire class. You will notice that the methods are overrides of the abstract methods. This is that they can have overrides in classes that inherit from **AnimatedSprite**.

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

namespace New2DRPG.SpriteClasses
{
    public enum AnimationKey { Up, Down, Left, Right };

    public class AnimatedSprite : Sprite
    {
        List<Animation> animations = new List<Animation>();
        AnimationKey currentAnimation;
        bool isAnimating;
        float speed = 3.0f;

        public AnimatedSprite(Game game, Texture2D texture, List<Animation>
animations)
            : base(game, texture)
        {
            spriteBatch = Game1.TileSpriteBatch;
            this.animations = animations;
            currentAnimation = AnimationKey.Down;
            isAnimating = false;
            width = animations[(int)currentAnimation].FrameWidth;
            height = animations[(int)currentAnimation].FrameHeight;
            center = new Vector2(width / 2, height / 2);
        }

        public float Speed
        {
            get { return speed; }
            set
            {
                speed = MathHelper.Clamp(value, 0.1f, 10f);
            }
        }

        public bool IsAnimating
        {
            get { return isAnimating; }
            set { isAnimating = value; }
        }

        public AnimationKey CurrentAnimation
        {
            get { return currentAnimation; }
            set { currentAnimation = value; }
        }

        public Rectangle CurrentRectangle
        {
            get
            {
                return animations[(int)currentAnimation].CurrentFrameRect;
            }
```

```
        }

        public override void Update(GameTime gameTime)
        {
            if (isAnimating)
                animations[(int)currentAnimation].Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
                SpriteSortMode.Deferred,
                SaveStateMode.None,
                Game1.Camera.TransformMatrix);

            spriteBatch.Draw(
                texture,
                Position,
                animations[(int)currentAnimation].CurrentFrameRect,
                Color.White);

            spriteBatch.End();
        }

        public void LockToMap()
        {
            if (position.X < 0)
                position.X = 0;
            if (position.Y < 0)
                position.Y = 0;
            if (position.X + width > TileMapComponent.WidthInPixels)
                position.X = TileMapComponent.WidthInPixels - width;
            if (position.Y + height > TileMapComponent.HeightInPixels)
                position.Y = TileMapComponent.HeightInPixels - height;

        }
    }
}
```

The **ItemSprite** class had to be changed as well because it inherited directly from the **Sprite** class. For that class there was a call to **base.Draw** for drawing the items and there was no **Update** method. The update method doesn't do anything but it is there in case it maybe needed down the road. It was a short class so the code for the entire class follows

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

namespace New2DRPG.SpriteClasses
{
    class ItemSprite : Sprite
    {
        Rectangle location;
```

```
        public ItemSprite(Game game, Texture2D texture, Vector2 position)
            : base(game, texture)
        {
            spriteBatch = Game1.TileSpriteBatch;
            this.position = new Vector2(
                position.X * TileEngine.TileWidth,
                position.Y * TileEngine.TileHeight);
            location = new Rectangle((int)this.position.X,
                    (int)this.position.Y,
                    TileEngine.TileWidth,
                    TileEngine.TileHeight);
        }

        public override void Update(GameTime gameTime)
        {
        }

        public override void Draw(GameTime gameTime)
        {
            spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
                SpriteSortMode.Deferred,
                SaveStateMode.None,
                Game1.Camera.TransformMatrix);
            spriteBatch.Draw(texture,
                    location,
                    Color.White);
            spriteBatch.End();
        }

    }
}
```

The **NPC** and **Monster** classes did not have to be modified because they inherited from the **AnimatedSprite** class. The **Chest** class has an **ItemSprite** field and didn't inherit from **ItemSprite** so it did not have to be changed either. Those are all of the changes that were made to those classes. You should be able to compile and run your game like normal.

The first manager class that I will make is a class that manages the items and chests in the game. I decided to merge them because they are so closely interlinked. To do that I made a class called **ItemManager** that will handle all of the items in the game and the chests in the game. It will trigger an event when the player collides with a chest. The first thing to do is to add the **ItemClasses** folder add a new class called **ItemManager**. There is a lot of code in the **ItemManager** class. To make adding the code for the class easier you can copy and paste the code for the **CreateChests**, **ReadItems**, **ReadWeapon**, **ReadArmor**, and **ReadShield** methods from the **Game1** class to the **ItemManager** class. I've included those methods in the code for the new class. You will have to change **this** in the **CreateChests** method to **Game** when creating the chests.

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

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Content;
```

```
namespace New2DRPG.ItemClasses
{
    public class ChestEventArgs : EventArgs
    {
        Chest chest;

        public ChestEventArgs(Chest chest)
        {
            Chest = chest;
        }

        public Chest Chest
        {
            get { return chest; }
            private set { chest = value; }
        }
    }

    public class ItemManager : DrawableGameComponent
    {
        public event ChestCollisionEventHandler ChestCollision;
        public delegate void ChestCollisionEventHandler(object sender,
ChestEventArgs e);

        List<Chest> chests = new List<Chest>();

        Dictionary<string, Weapon> weapons = new Dictionary<string, Weapon>();
        Dictionary<string, Armor> armors = new Dictionary<string, Armor>();
        Dictionary<string, Shield> shields = new Dictionary<string, Shield>();
        Random random = new Random();
        ContentManager Content;

        public ChestManager(Game game)
            : base(game)
        {
            Content =
                (ContentManager)Game.Services.GetService(typeof(ContentManager));
            Initialize();
        }

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

        protected override void LoadContent()
        {
            ReadItems(@"Content\Items\items.its");
            CreateChests();
        }

        public override void Update(GameTime gameTime)
        {
            foreach (Chest chest in chests)
                chest.Update(gameTime);
        }
```

```
public override void Draw(GameTime gameTime)
{
    foreach (Chest chest in chests)
        chest.Draw(gameTime);
}

public void CheckChestCollision(Vector2 origin)
{
    for (int i = 0; i < chests.Count; i++)
    {
        float distance = Vector2.Distance(
            chests[i].Origin,
            origin);

        if (distance < Chest.CollisionRadius)
        {
            if (ChestCollision != null)
            {
                ChestEventArgs e = new ChestEventArgs(chests[i]);
                chests.RemoveAt(i);
                ChestCollision(this, e);
            }
        }
    }
}

private void CreateChests()
{
    Chest tempChest;

    for (int i = 0; i < 2; i++)
    {
        tempChest = new Chest(
            Game,
            Content.Load<Texture2D>(@"Items\chest"),
            new Vector2(random.Next(3, 3 + 5), random.Next(3, 3 + 5)),
            random.Next(100),
            random.Next(100, 200),
            null);
        chests.Add(tempChest);
    }

    List<BaseItem> items = new List<BaseItem>();

    items.Add(weapons["dagger"]);
    items.Add(armors["leather armor"]);

    tempChest = new Chest(
        Game,
        Content.Load<Texture2D>(@"Items\chest"),
        new Vector2(random.Next(5, 11), random.Next(5, 11)),
        random.Next(100),
        random.Next(100, 200),
        items);

    chests.Add(tempChest);

    List<BaseItem> items2 = new List<BaseItem>();
    items2.Add(shields["buckler"]);
```

```
        tempChest = new Chest(
            Game,
            Content.Load<Texture2D>(@"Items\chest"),
            new Vector2(random.Next(5, 11), random.Next(5, 11)),
            items2);

        chests.Add(tempChest);
    }

    private void ReadItems(string filename)
    {
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.Load(filename);

        XmlNode root = xmlDoc.FirstChild;

        weapons.Clear();
        armors.Clear();
        shields.Clear();

        foreach (XmlNode node in root.ChildNodes)
        {
            if (node.Name == "Weapons")
                foreach (XmlNode wpn in node.ChildNodes)
                    ReadWeapon(wpn);
            if (node.Name == "Armors")
                foreach (XmlNode arm in node.ChildNodes)
                    ReadArmor(arm);
            if (node.Name == "Shields")
                foreach (XmlNode sld in node.ChildNodes)
                    ReadShield(sld);
        }
    }

    private void ReadWeapon(XmlNode wpn)
    {
        string name;
        int price;
        int weight;
        ItemSize size;
        Hands hands;
        int attackValue;
        int attackBonus;

        XmlNode node = wpn.FirstChild;
        if (node.Name != "Name")
            throw new Exception("Illegal file format!");
        name = node.Attributes[0].Value;

        node = node.NextSibling;
        if (node.Name != "Price")
            throw new Exception("Illegal file format!");
        price = int.Parse(node.Attributes[0].Value);

        node = node.NextSibling;
        if (node.Name != "Weight")
            throw new Exception("Illegal file format!");
        weight = int.Parse(node.Attributes[0].Value);
```

```
            node = node.NextSibling;
            if (node.Name != "Size")
                throw new Exception("Illegal file format!");
            size = (ItemSize)Enum.Parse(typeof(ItemSize),
node.Attributes[0].Value);

            node = node.NextSibling;
            if (node.Name != "Hands")
                throw new Exception("Illegal file format!");
            hands = (Hands)Enum.Parse(typeof(Hands), node.Attributes[0].Value);

            node = node.NextSibling;
            if (node.Name != "Attack")
                throw new Exception("Illegal file format!");
            attackValue = int.Parse(node.Attributes[0].Value);
            attackBonus = int.Parse(node.Attributes[1].Value);

            Weapon weapon = new Weapon(
                    name,
                    price,
                    weight,
                    size,
                    hands,
                    attackValue,
                    attackBonus);
            weapons.Add(name, weapon);
        }

        private void ReadArmor(XmlNode arm)
        {
            string name;
            int price;
            int weight;
            ItemSize size;
            int defenseValue;
            int defenseBonus;

            XmlNode node = arm.FirstChild;
            if (node.Name != "Name")
                throw new Exception("Illegal file format!");
            name = node.Attributes[0].Value;

            node = node.NextSibling;
            if (node.Name != "Price")
                throw new Exception("Illegal file format!");
            price = int.Parse(node.Attributes[0].Value);

            node = node.NextSibling;
            if (node.Name != "Weight")
                throw new Exception("Illegal file format!");
            weight = int.Parse(node.Attributes[0].Value);

            node = node.NextSibling;
            if (node.Name != "Size")
                throw new Exception("Illegal file format!");
            size = (ItemSize)Enum.Parse(typeof(ItemSize),
node.Attributes[0].Value);
```

```
            node = node.NextSibling;
            if (node.Name != "Defense")
                throw new Exception("Illegal file format!");
            defenseValue = int.Parse(node.Attributes[0].Value);
            defenseBonus = int.Parse(node.Attributes[1].Value);

            Armor armr = new Armor(
                    name,
                    price,
                    weight,
                    size,
                    defenseValue,
                    defenseBonus);
            armors.Add(name, armr);
        }

        private void ReadShield(XmlNode sld)
        {
            string name;
            int price;
            int weight;
            ItemSize size;
            int defenseValue;
            int defenseBonus;

            XmlNode node = sld.FirstChild;
            if (node.Name != "Name")
                throw new Exception("Illegal file format!");
            name = node.Attributes[0].Value;

            node = node.NextSibling;
            if (node.Name != "Price")
                throw new Exception("Illegal file format!");
            price = int.Parse(node.Attributes[0].Value);

            node = node.NextSibling;
            if (node.Name != "Weight")
                throw new Exception("Illegal file format!");
            weight = int.Parse(node.Attributes[0].Value);

            node = node.NextSibling;
            if (node.Name != "Size")
                throw new Exception("Illegal file format!");
            size = (ItemSize)Enum.Parse(typeof(ItemSize),
node.Attributes[0].Value);

            node = node.NextSibling;
            if (node.Name != "Defense")
                throw new Exception("Illegal file format!");
            defenseValue = int.Parse(node.Attributes[0].Value);
            defenseBonus = int.Parse(node.Attributes[1].Value);

            Shield shld = new Shield(
                    name,
                    price,
                    weight,
                    size,
                    defenseValue,
                    defenseBonus);
```

```
            shields.Add(name, shld);
        }
    }
}
```

There are using statements for the XNA and XML classes that are needed in the class. You will see that there is a second class inside this class called **ChestEventArgs** that derives from the **EventArgs** class. I will be using this class for creating the event that will be fired when the player collides with a chest. It has a field, constructor, and property. The field will be the chest that the player collides with and the constructor will set the field. The get part returns the chest and the set part is private and is used to set the field.

I inherited the class from **DrawableGameComponent** so that it will have both **Update** and **Draw** methods as well as other methods available to game components. There is an event called **ChestCollision** and a delegate for that event in the class. Events are good ways of having communication between different classes. This event will be triggered when the player collides with a chest and the event is subscribed to. If the event isn't subscribed to nothing will happen. The delegate describes the method that will be called when the event is triggered. There are fields from the **Game1** class. They are for all of the chests in the game and fields for the weapons, armor, and shields. There is also a **Random** field for generating random numbers and a **ContentManager** field for loading in content.

The constructor for the class retrieves the current **ContentManager** object and calls the **Initialize** method. The **Initiliaze** method automatically calls the **LoadContent** method with the call to **base.Initialize**. The **LoadContent** method then calls the **ReadItems** method to read in the items for the game and then **CreateChests** to create the chests for the game. The **Update** and **Draw** methods just loop through all of the chests and call their **Update** and **Draw** methods.

There is one method that you have seen before but not quite. That is the **CheckChestCollision** method. This method is called to check to see if the player has collided with a chest. If the player has collided with a chest event handler code will be triggered, if the event is subscribed to. The method takes a **Vector2** as a parameter that is the origin of the player's sprite.

The method loops through all of the chests in the game. It gets the distance between the origin of the player and the origin of the chest and compares it to the collision radius of the chest class. What happens if that condition is true is I check to see if the event has been subscribed to by checking to see if it is not null. I then create an instance of **ChestEventArgs** to hold the chest that triggered to collision. Like before I remove the chest for the **List<Chest>** so it is no longer part of the game and then I call the event handler passing in the sender, which is the current instance, and the instance of the **ChestEventArgs**.

Before getting to implementing this in the game, I want to make one quick change to the **PlayerComponent**. What I want to do is to add a get only property to get the origin of the player. Add the following property to the **PlayerComponent** class.

```
public Vector2 Origin
{
    get { return sprite.Origin; }
}
```

Now it is time to implement this manager in the game. The first thing you can do is go to the fields of the class. Find the **chests**, **weapons**, **armors,** and **shields** fields. Replace them with the following field.

```
ItemManager itemManager;
```

This will break a few things but don't worry about it. You can remove the **CreateChests**, **ReadItems**, **ReadWeapon**, **ReadArmor**, and **ReadShield** methods from the **Game1** class. You don't need them any more. The next thing do is change the **LoadContent** method to the following.

```
protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);
    tileSpriteBatch = new SpriteBatch(GraphicsDevice);

    Services.AddService(typeof(SpriteBatch), spriteBatch);
    Services.AddService(typeof(ContentManager), Content);

    dialog = new DialogComponent(this);
    Components.Add(dialog);

    normalFont = Content.Load<SpriteFont>("normal");

    LoadGameScreens();

    CreateAnimations();

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

    script = ReadScript(@"Content\script1.script");

    LoadPlayerSprites();
    CreatePlayerAnimations();

    CreateNPCS();
    CreateMonsters();
    CreateItemManager();
}
```

Instead of the **LoadConent** method calling the **CreateChests** method, it now calls a method called **CreateItemManager**. The **CreateItemManager** method creates the **ItemManager** instance and wires the **ChestCollision** event handler. This is the code for the **CreateItemManager** method and the event handler.

```
private void CreateItemManager()
{
    itemManager = new ItemManager(this);
    itemManager.ChestCollision +=
        new ItemManager.ChestCollisionEventHandler(chestManager_ChestCollision);
}

void chestManager_ChestCollision(object sender, ChestEventArgs e)
{
    activeScreen.Enabled = false;
```

```
    activeScreen = treasureScreen;
    treasureScreen.Show(e.Chest);
}
```

The code for the event handler is created the same was as the code wiring events in a Windows forms application. You type += to associate an event handler and then you can press tab twice to generate a method stub for the event handler. The code for the event handler should look very familiar. All it does is hide the active screen, set the active screen to be the treasure screen and call the show method passing in the chest that was passed to the **ChestEventArgs** when the event was fired.

What you need to do now is change the **HandlePlayerInput** method. Before that method would loop through all of the chests, call their **Update** methods, and check for collisions. You can now just call the **Update** and **CheckChestCollision** methods passing in the **GameTime** parameter and the origin of the player's sprite respectively. This is the new code for the **HandlePlayerInput** method.

```
private void HandlePlayerInput(GameTime gameTime)
{
    player.Update(gameTime);

    if (!inDialog)
    {
        foreach (NPC npc in npcs)
            npc.Update(gameTime);

        if (CheckAttackRadius(gameTime))
            return;

        itemManager.Update(gameTime);
        itemManager.CheckChestCollision(player.Origin);

        if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
            CheckSpeakingRadius();
    }
    if (!inDialog)
        HandlePlayerMovement();
}
```

The last change will be in the **Draw** method of the **Game1** class. Instead of looping through the **List<Chest>** to draw the chests you can now just call the **Draw** method of the **ItemManager** class to draw all of the chests. This is the new **Draw** method.

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
    base.Draw(gameTime);
    if (activeScreen == actionScreen || activeScreen == quitActionScreen)
    {
        player.Draw(gameTime);
        foreach (NPC sprite in npcs)
            sprite.Draw(gameTime);
        foreach (Monster monster in monsters)
        {
            if (!monster.InCombat)
                monster.Draw(gameTime);
        }
```

```
            itemManager.Draw(gameTime);
        }
        spriteBatch.End();
    }
```

Well, that is it for this tutorial. I had thought about adding in components for managing the NPCs in the game and the monsters in the game in this tutorial. That ended up being a bigger task than I had originally thought it would be. I will be getting to that in the next tutorial more than likely.

I will be working on these tutorials and I will be adding in more and more functionality to the game that you would expect to find in a role playing game. I encourage you to keep either visiting my site http://xnagpa.net or my blog, XNA Game Programming Adventures Blog for the latest news on my tutorials.