Creating a Role Playing Game with XNA Game Studio
Part 47
Picking Up Items - 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/New2DRPG46.zip You can download the graphics from this link: Graphics.zip

What I will be working on is this tutorial is having the player being able to pick up items. This is the first part in which I will set up a few classes that we will need and do a little bit of drawing with XNA. To handle picking up items I decided to use chests to hold the items. It is easier to draw the chests and trigger the player picking them up than drawing all sorts of items and having the player picking them up. With chests you can also put coins in as well.

The first thing I did was find an image better for chests than the one I created. You can download the image from chest.zip on my web site. After you download and unzip the file add it to the **Items** folder in the **Content** folder. You will get a message saying that the file already exists, go a head and click **OK** to apply the change.

I am going to make just a small change to the **ItemSprite** class. What I did is I created a **Rectangle** field in the class to hold the location of the chest on the map. Remember that this location is measured in tiles, not pixels, and that drawing the chest is done with the **SpriteBatch** object for the tile engine. I'm also going to set the **position** vector to be the position of the sprite in pixels. The reason I'm changing **position** is I am going to use the same method of determining when the player is in speaking range of an NPC as with the items. This is the new code for the **ItemSprite** class.

```
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 Draw(GameTime gameTime)
        {
            base.Draw(gameTime);

            spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
                SpriteSortMode.Deferred,
                SaveStateMode.None,
                Game1.Camera.TransformMatrix);
            spriteBatch.Draw(texture,
                    location,
                    Color.White);
            spriteBatch.End();
        }

    }
}
```

The next thing to do is to add a new folder to the game project. In this folder I will be adding classes related to items for the game. So, right click your game project and select **Add** and then **New Folder**. Name this folder **ItemClasses**. To this folder add a new class called **Chest**. This class will hold the items and gold for the player to pick up. I will explain the code after you have read it.

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

namespace New2DRPG.ItemClasses
{
    class Chest
    {
        static Random random = new Random();
        ItemSprite sprite;
        static float collisionRadius = 24;
        int goldMinimum;
        int goldMaximum;

        public Chest(Game game, Texture2D texture, Vector2 position)
        {
            sprite = new ItemSprite(game, texture, position);
        }

        public static float CollisionRadius
        {
            get { return collisionRadius; }
        }

        public Vector2 Position
```

```
        {
            get { return sprite.Position; }
        }

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

        public int Gold
        {
            get
            {
                if (goldMinimum == goldMaximum)
                    return goldMinimum;
                else
                    return Chest.random.Next(goldMinimum, goldMaximum + 1);
            }
        }

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

        public void Draw(GameTime gameTime)
        {
            sprite.Draw(gameTime);
        }
    }
}
```

There are using statements for both the XNA framework and for the XNA framework graphics classes. The one for the graphics classes is because the chests are going to be responsible for drawing themselves and have an **ItemSprite** field. To create the **ItemSprite** you need to pass it a **Texture2D** for the image. Since the class has an **ItemSprite** field I need to also add a using statement for the sprite classes.

The fist field might confuse you a little. I added in a static object of the **Random** class called **random**. There is a good reason for making it static. Chests will be able to hold a fixed or a random amount of gold. To calculate the random amount of gold I will need an instance of the **Random** class. Instead of a separate object in each class, having the instance static allows all of the objects of the **Chest** class to share the same instance. This will save a little memory which is a good thing. As I mentioned there is a **ItemSprite** field so the chest will be able to draw itself on the map. As I mentioned I will be using the same method of detecting if the player collides with a chest as with colliding with sprites, using a collision radius. I decided to have this shared among all of the chests in the game so it is static as well. The next two fields will be used for determining the minimum and maximum amount of gold a chest can hold. I will use these fields to generate the gold in each chest.

The constructor for the class takes three parameters at the moment. The first is a **Game** object that the **ItemSprite** class needs. The other two are also needed for the **ItemSprite** class. The **Texture2D** for the sprite and the position of the chest. This is measured in tiles, not pixels. When the sprite is created it will be converted to pixels.

There are four public get only properties for this class. The first one is static and it returns the **collisionRadius** field. The next two are **Vector2** properties that return the **Position** and **Origin** properties of the **ItemSprite** class. The last property returns the gold the chest contains. If the values of **goldMinimum** and **goldMaximum** are the same the chest holds just that amount of gold. Setting both of these to zero will have the chest hold no gold at all. When you are creating a chest and you want it to have a range of gold you set **goldMinimum** to the lowest value in the range and **goldMaximum** to the highest value in the range. I will handle setting these values in the next tutorial.

There are two public methods in the class: **Update** and **Draw**. The **Update** method calls the **Update** method of the **ItemSprite** field. The **Draw** method does the same for the **Draw** method of the **ItemSprite** field.

The next thing to do is to add some chests to the game. They will be added in the **Game1** class. The first thing to do is to add a using statement to the game for the **ItemClasses** namespace. Add the following using statement with the other using statements.

```
using New2DRPG.ItemClasses;
```

The next thing to do is to add a field to the class to hold the chests in the game. I will use a **List<Chests>** to hold all of the chests in the game. Add this field to the **Game1** class above the constructor of the class.

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

The next thing to do is to actually create some chests. What I did as at the end of the **LoadContent** method is call a new method that I wrote called **CreateChests**. I will explain the new method after you have read the code. This is the new code for the **LoadContent** method and the **CreateChests** method.

```
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();
    CreateChests();
}

private void CreateChests()
{
    for (int i = 0; i < 2; i++)
    {
        Chest tempChest = new Chest(
            this,
            Content.Load<Texture2D>(@"Items\chest"),
            new Vector2(random.Next(3, 3 + 5), random.Next(3, 3 + 5)));
        chests.Add(tempChest);
    }
}
```

The **CreateChests** method in a for loop creates an instance of the **Chest** class passing in the current **Game** object, the **Texture2D** of the chest, and a random **Vector2**. It then adds the new chest to the **List<Chest>**.

Drawing the chests will happen in the **Draw** method. What I did was, in a foreach loop, is loop through all of the chests and call their **Draw** method after the loop that draws the monsters. This is the updated **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);
        }
        foreach (Chest chest in chests)
            chest.Draw(gameTime);
    }
    spriteBatch.End();
}
```

In the **HandlePlayerInput** method is where I handled checking to see if the player's sprite collides with a chest. What I did was in an if statement call a method that I wrote **CheckPickupRadius** like the other methods that check for collisions that will return true if the player collides with a chest. This is the 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;

        if (CheckPickupRadius(gameTime))
            return;

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

For now the **CheckPickupRadius** method just loops through all of the chests in the game, calls their **Update** method, and returns false. This is the code for the **CheckPickupRadius** method.

```
private bool CheckPickupRadius(GameTime gameTime)
{
    foreach (Chest chest in chests)
        chest.Update(gameTime);

    return false;
}
```

Well, that is it for this tutorial. I know that it is a little short but I didn't want to do too much in one tutorial. In the next tutorial I will continue on with picking up items. 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.