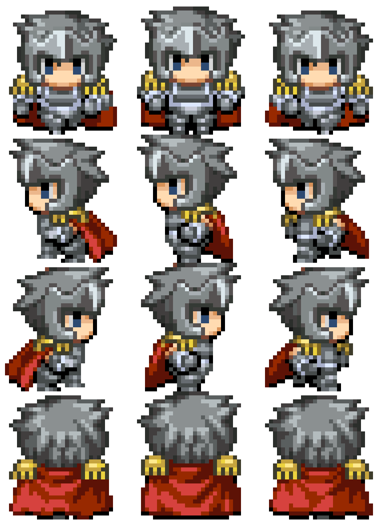
Creating a Role Playing Game with XNA Game Studio 3.0
Part 43
Different Sprites

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/New2DRPG42.zip You can download the graphics from this link: Graphics.zip

In this tutorial I will be adding in different sprites for each of the character classes and genders. I made a few sprites, with the help of an online sprite generating program, to add to the game. They aren't quite like the ones from Last Guardian that I've been using so far. They are not in the same format with the up animation first and on the same line. I will be able to handle this when I load them in. This is a sample of what the sprites look like.



Since there are four classes and two genders there are eight sprites in total. You can download the eight sprites from this link. http://xna.jtmbooks.com/Downloads/playersprites.zip Extract the sprites to a directory and remember where you extracted them. The names of the sprites are: **femalefighter.png**, **femalepriest.png**, **femalethief.png**, **femalewizard.png**, **malefighter.png**, **malepriest.png**, **malethief.png** and **malewizard.png**. Add all of these sprites to the **Sprites** folder in the **Content** folder of your game.

Since these sprites are different than the other sprites from the game Last Guardian you will need to create new animations for them. You will also need to load in their textures for use. What I am going to do is add a **List<Animation>** to hold the animations and a two dimensional array of

Texture2D to hold the textures of the sprites. Add the following two fields to the code for **Game1** near the fields for the **PlayerCharacter** component.

```
List<Animation> playerAnimations = new List<Animation>();
Texture2D[,] playerTextures = new Texture2D[4, 2];
```

I will handle the loading of the sprites and the creation of their animations in the **LoadContent** method of the game. From the **LoadContent** method I will call two methods that I wrote called :**LoadPlayerSprites** and **CreatePlayerAnimations**. The first will load the textures of the sprites into the array and the second will create the animations. I also added a third method: **ClonePlayerAnimations** to clone the animations. I will explain the three new methods after you have seen the code for them. This is the code for those four 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");
    LoadCreatePCScreen();
    LoadStartScreen();
    LoadHelpScreen();
    LoadActionScreen();
    LoadQuitPopUpScreen();
    LoadGenderPopUpScreen();
    LoadClassPopUpScreen();
    LoadDifficultyPopUpScreen();
    LoadNameInputScreen();
    LoadCreditScreen();
    LoadIntroScreen();
    LoadViewCharacterScreen();
    LoadQuitActionScreen();
    LoadCombatScreen();
    creditScreen.Hide();
    startScreen.Hide();
    helpScreen.Hide();
    createPCScreen.Hide();

    activeScreen = introScreen;
    activeScreen.Show();

    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();
}

private void LoadPlayerSprites()
{
    playerTextures[0, 0] = Content.Load<Texture2D>(@"Sprites\malefighter");
    playerTextures[1, 0] = Content.Load<Texture2D>(@"Sprites\malewizard");
    playerTextures[2, 0] = Content.Load<Texture2D>(@"Sprites\malepriest");
    playerTextures[3, 0] = Content.Load<Texture2D>(@"Sprites\malethief");

    playerTextures[0, 1] = Content.Load<Texture2D>(@"Sprites\femalefighter");
    playerTextures[1, 1] = Content.Load<Texture2D>(@"Sprites\femalewizard");
    playerTextures[2, 1] = Content.Load<Texture2D>(@"Sprites\femalepriest");
    playerTextures[3, 1] = Content.Load<Texture2D>(@"Sprites\femalethief");
}

private void CreatePlayerAnimations()
{
    Animation tempAnimation = new Animation(3, 64, 64, 0, 192);
    playerAnimations.Add(tempAnimation);

    tempAnimation = new Animation(3, 64, 64, 0, 0);
    playerAnimations.Add(tempAnimation);

    tempAnimation = new Animation(3, 64, 64, 0, 64);
    playerAnimations.Add(tempAnimation);

    tempAnimation = new Animation(3, 64, 64, 0, 128);
    playerAnimations.Add(tempAnimation);
}

private List<Animation> ClonePlayerAnimations()
{
    List<Animation> newAnimation = new List<Animation>();

    foreach (Animation a in playerAnimations)
    {
        Animation clonedAnimation = (Animation)a.Clone();
        newAnimation.Add(clonedAnimation);
    }

    return newAnimation;
}
```

The **LoadPlayerSprites** method just uses the **Content.Load** method to load in the textures for the sprites. The order in which they are loaded is important however. You will notice that the male sprites are loaded in before the female sprites. The sprites are loaded in with the fighter first, wizard second, priest third and thief last. The reason for this is that for the gender of the character I used a bool field. If it is false the player character is male and female if it is true. The character classes are listed: **Fighter**, **Wizard**, **Priest** and **Thief**. You will see why more when I change the **CreatePlayerCharacter** method to have the sprite for player character match what was selected in the character generator.

In the **CreatePlayerAnimations** method I create the actual animation frames for the sprites. The way it was done is a little different that the other sprites. Instead of editing the images of the sprites I decided it was easier to handle the differences in the code. The **CreatePlayerAnimations** method first creates an instance of the **Animation** class passing in the values: **3**, **64**, **64**, **0**, and **192**. The first one is the number of frames for the animation. There are 3 frames for each animation. The sprites are all 64 pixels by 64 pixels so those will all be 64 and 64 as well. Since the sprites all line up vertically the forth parameter will always be 0. It is the last parameter that is interesting. The animation of the sprite moving up is in the forth row so I got 192 by multiplying the height of the sprite by 3. The down animation is the first animation so I multiplied 64 by 0 to get 0. Left and right are second and third so to get those values I multiplied 64 by 1 and 3 to get 64 and 128. Since the sprite sheet is zero based I subtracted the row number by 1 to get the appropriate row. Each of the animations is added to the **playerAnimations** field.

The **ClonePlayerAnimations** method is basically the same as the **CloneAnimations** method. The difference is that in the foreach loop instead of looping through the **animations** field I loop through the **playerAnimations** field, clone them, add them to a **List<Animation>** and return that list.

Next I had to send the proper sprite for the player's choice of character when making the sprite for the player's character. That was done in the **CreatePlayerCharacter** method, and in the **HandleStartScreenInput** method because at the moment I do not allow for the saving and loading of games. When you choose the second menu item it just starts the game with a default character. This is the code for the **CreatePlayerCharacter** method.

```
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);
    }

    if (createPCScreen.CharacterGender)
    {
        playerSprite = new AnimatedSprite(this,
            playerTextures[(int)createPCScreen.CharacterClass, 1],
            ClonePlayerAnimations());
    }
    else
    {
        playerSprite = new AnimatedSprite(this,
            playerTextures[(int)createPCScreen.CharacterClass, 0],
            ClonePlayerAnimations());
    }

    playerSprite.CurrentAnimation = AnimationKey.Down;
    player = new PlayerComponent(this, playerSprite, playerCharacter);
    player.Hide();
}
```

What is different in this method is where I created the sprite to pass to the **PlayerComponent** class. There is now an if statement that checks to see what the **CharacterGender** property is in the **CreatePCScreen**. To find out what class the player character is I use the **CharacterClass** property of the **CreatePCScreen** class and cast that to an integer. This is where the order that I loaded the sprites in becomes important. I loaded the sprites in the same order as the selections in the enum that holds the different classes the player can choose from. Since the **CharacterGender** property is true for female and false for male if the **CharacterGender** is true for the second index of the array I use 1 and if it is false I used 0. I also call the **CloneAnimations** method to clone the animations for the sprite when I create the sprite.

I did pretty much the same thing in the **HandleStartScreenInput** method. Since the default character is a male fighter I used the texture at index [0, 0] when creating the sprite. This is the code for that method.

```
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);
                playerSprite = new AnimatedSprite(this,
                    playerTextures[0, 0],
                    ClonePlayerAnimations());
                playerSprite.CurrentAnimation = AnimationKey.Down;
                player = new PlayerComponent(this, playerSprite, playerCharacter);
                activeScreen = actionScreen;
                player.Show();
                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;
        }
    }
}
```

Another thing that I thought would be nice is in the character generator to display a picture of the player's choice of character. I have the sprites so I decided to put a picture of the first frame of the down animation above the name, class and difficulty level of the player's choice. All of this will be done in the **CreatePCScreen** class. The first thing to do is to load in the textures for the sprite. To load them in you will need an array for the textures like in the game. Add the following field to the **CreatePCScreen** class just above the constructor.

```
Texture2D[,] playerTextures = new Texture2D[4, 2];
```

Now you can add in the code to load the images. What I did was copy and paste the method from the game and call it from the **LoadConent** method. This is the code I added to the **CreatePCScreen** for loading the content for the game.

```
protected override void LoadContent()
{
  background = Content.Load<Texture2D>(@"Backgrounds\createpcscreen");
  buttonImage = Content.Load<Texture2D>(@"GUI\buttonbackground");
  spriteFont = Content.Load<SpriteFont>("normal");
  LoadPlayerSprites();
  base.LoadContent();
}

private void LoadPlayerSprites()
{
  playerTextures[0, 0] = Content.Load<Texture2D>(@"Sprites\malefighter");
  playerTextures[1, 0] = Content.Load<Texture2D>(@"Sprites\malewizard");
  playerTextures[2, 0] = Content.Load<Texture2D>(@"Sprites\malepriest");
  playerTextures[3, 0] = Content.Load<Texture2D>(@"Sprites\malethief");

  playerTextures[0, 1] = Content.Load<Texture2D>(@"Sprites\femalefighter");
  playerTextures[1, 1] = Content.Load<Texture2D>(@"Sprites\femalewizard");
  playerTextures[2, 1] = Content.Load<Texture2D>(@"Sprites\femalepriest");
  playerTextures[3, 1] = Content.Load<Texture2D>(@"Sprites\femalethief");
}
```

All that is left is to draw the sprite to show the player what the character they have chosen looks like. That will of course be done in the **Draw** method. I used a method similar to what I had done when I was creating the sprite in the game. I will give you the code and then explain it.

```
public override void Draw(GameTime gameTime)
{
    Vector2 position = new Vector2();
    string characterString;

    characterString = name + " the ";

    base.Draw(gameTime);

    characterString += classNames[className];
    Vector2 stringSize = spriteFont.MeasureString(characterString);
    position.X = (screenWidth - stringSize.X) / 2;
    position.Y = 280;

    spriteBatch.DrawString(spriteFont,
        characterString,
        position + Vector2.One * 3,
        Color.Black);

    spriteBatch.DrawString(spriteFont,
        characterString,
        position,
        Color.White);

    characterString = "Playing in " + difficultyLevels[difficultyIndex];
    characterString += " mode";
    stringSize = spriteFont.MeasureString(characterString);
    position.X = (screenWidth - stringSize.X) / 2;
    position.Y += spriteFont.LineSpacing;

    spriteBatch.DrawString(spriteFont,
        characterString,
        position + Vector2.One * 3,
        Color.Black);

    spriteBatch.DrawString(spriteFont,
        characterString,
        position,
        Color.White);

    position.Y = 200;
    position.X = (screenWidth - 64) / 2;

    if (gender)
    {
        spriteBatch.Draw(playerTextures[(int)characterClass, 1],
            position,
            new Rectangle(0, 0, 64, 64),
            Color.White);
    }
    else
    {
        spriteBatch.Draw(playerTextures[(int)characterClass, 0],
            position,
            new Rectangle(0, 0, 64, 64),
            Color.White);
    }
}
```

What I had done is after drawing everything else is set the **position** variable to where I wanted to draw the sprite. For the **Y** property I set it to 200 pixels. This value was found by trial and error. I had just found a position that looked appropriate. For the **X** property I took the width of the screen, subtracted the width of the sprite, which is 64 pixels, and divided the result by two to center it in the middle of the screen horizontally. I again used an if statement to determine the gender of the character. If the gender was true I draw a female sprite passing using 1 for the second index and male otherwise passing using 0 for the second index. To find the first index I cast the **characterClass** field to an integer. The overload of the **Draw** method I used for the **SpriteBatch** class was the overload that takes as its parameters: the **Texture2D** of the image you want to draw, the position you want to draw it as a **Vector2**, the source **Rectangle** of the image, and the tint **Color**. For the source rectangle I used a rectangle that started at (0, 0) and was 64 pixels wide and 64 pixels high.

This was a shorter tutorial than most lately but it was adding something that I felt was missing from the game. Like I said in the previous tutorial, the game is getting more functional in terms of XNA but there is still much missing relating to game mechanics. I hope to get to that soon. I will be starting the next part of Eyes of the Dragon shortly. 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.