Creating a Role Playing Game with XNA Game Studio 3.0
Part 11
Creating a Textbox Control

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: Eyes of the Dragon - Version 10 You can download the graphics for the these tutorials at this link: Graphics.zip

Again, I did quite a bit of programming on Eyes of the Dragon, the name I have given to the game. To start with I created a textbox control for use in the game. Then I created a simple pop up screen that had a textbox control on it. I then change the character generator to use the new pop up screen for getting the character's name instead of getting it from a list of names.

To get started, go a head and load the last version of the project. Before I go any farther you will want to download the graphics from the graphics file listed above. You will want to add the **textbox.png** and **cursor.png** files to the **GUI** subfolder in the **Content** folder.

To get started, you will want to add a new **GameComponent** to the **CoreComponents** folder called **Textbox**. As I always do, I will give you the new code and then explain what I've done. This is the code for the **Textbox GameComponent**.

```
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
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class Textbox : Microsoft.Xna.Framework.DrawableGameComponent
    {
        Texture2D textboxTexture;
        Texture2D cursor;

        SpriteFont spriteFont;
        SpriteBatch spriteBatch;
        ContentManager Content;
```

```
string text;

Keys[] keysToCheck = new Keys[] {
    Keys.A, Keys.B, Keys.C, Keys.D, Keys.E,
    Keys.F, Keys.G, Keys.H, Keys.I, Keys.J,
    Keys.K, Keys.L, Keys.M, Keys.N, Keys.O,
    Keys.P, Keys.Q, Keys.R, Keys.S, Keys.T,
    Keys.U, Keys.V, Keys.W, Keys.X, Keys.Y,
    Keys.Z, Keys.Back, Keys.Space };

Vector2 cursorPosition;
Vector2 textPosition;
Vector2 textboxPosition;

TimeSpan blinkTime;
bool blink;

KeyboardState currentKeyboardState;
KeyboardState lastKeyboardState;

public Textbox(Game game, SpriteFont spriteFont)
    : base(game)
{
    spriteBatch =
        (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));

    Content =
        (ContentManager)Game.Services.GetService(typeof(ContentManager));

    this.spriteFont = spriteFont;

    textboxTexture = Content.Load<Texture2D>(@"GUI\textbox");
    cursor = Content.Load<Texture2D>(@"GUI\cursor");

    textboxPosition = new Vector2();
    cursorPosition = new Vector2(
        textboxPosition.X + 5,
        textboxPosition.Y + 5);
    textPosition = new Vector2(
        textboxPosition.X + 5,
        textboxPosition.Y + 5);
    blink = false;
    text = "";
}

public string Text
{
    get { return text; }
    set { text = value; }
}

public Vector2 Position
{
    get { return textboxPosition; }
    set
    {
        textboxPosition = value;
        SetTextPosition();
    }
```

```
}

private void SetTextPosition()
{
    cursorPosition = new Vector2(
        textboxPosition.X + 5,
        textboxPosition.Y + 5);
    textPosition = new Vector2(
        textboxPosition.X + 5,
        textboxPosition.Y + 5);
}

public int Height
{
    get { return textboxTexture.Height; }
}

public int Width
{
    get { return textboxTexture.Width; }
}

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

public override void Update(GameTime gameTime)
{
    currentKeyboardState = Keyboard.GetState();
    blinkTime += gameTime.ElapsedGameTime;
    if (blinkTime > TimeSpan.FromMilliseconds(500))
    {
        blink = !blink;
        blinkTime -= TimeSpan.FromMilliseconds(500);
    }
    foreach (Keys key in keysToCheck)
    {
        if (CheckKey(key))
        {
            AddKeyToText(key);
            break;
        }
    }
    base.Update(gameTime);
    Vector2 textSize = spriteFont.MeasureString(text);
    cursorPosition.X = textPosition.X + textSize.X;
    lastKeyboardState = currentKeyboardState;
}

private void AddKeyToText(Keys key)
{
    string newChar = "";

    if (text.Length >= 16 && key != Keys.Back)
        return;

    switch (key)
    {
```

```
case Keys.A:
    newChar += "a";
    break;
case Keys.B:
    newChar += "b";
    break;
case Keys.C:
    newChar += "c";
    break;
case Keys.D:
    newChar += "d";
    break;
case Keys.E:
    newChar += "e";
    break;
case Keys.F:
    newChar += "f";
    break;
case Keys.G:
    newChar += "g";
    break;
case Keys.H:
    newChar += "h";
    break;
case Keys.I:
    newChar += "i";
    break;
case Keys.J:
    newChar += "j";
    break;
case Keys.K:
    newChar += "k";
    break;
case Keys.L:
    newChar += "l";
    break;
case Keys.M:
    newChar += "m";
    break;
case Keys.N:
    newChar += "n";
    break;
case Keys.O:
    newChar += "o";
    break;
case Keys.P:
    newChar += "p";
    break;
case Keys.Q:
    newChar += "q";
    break;
case Keys.R:
    newChar += "r";
    break;
case Keys.S:
    newChar += "s";
    break;
case Keys.T:
    newChar += "t";
```

```
                break;
            case Keys.U:
                newChar += "u";
                break;
            case Keys.V:
                newChar += "v";
                break;
            case Keys.W:
                newChar += "w";
                break;
            case Keys.X:
                newChar += "x";
                break;
            case Keys.Y:
                newChar += "y";
                break;
            case Keys.Z:
                newChar += "z";
                break;
            case Keys.Space:
                newChar += " ";
                break;
            case Keys.Back:
                if (text.Length != 0)
                    text = text.Remove(text.Length - 1);
                return;
        }
        if (currentKeyboardState.IsKeyDown(Keys.RightShift) ||
            currentKeyboardState.IsKeyDown(Keys.LeftShift))
        {
            newChar = newChar.ToUpper();
        }
        text += newChar;
    }

    private bool CheckKey(Keys theKey)
    {
        return lastKeyboardState.IsKeyDown(theKey) &&
               currentKeyboardState.IsKeyUp(theKey);
    }

    public override void Draw(GameTime gameTime)
    {
        spriteBatch.Draw(textboxTexture, textboxPosition, Color.White);

        if (!blink)
            spriteBatch.Draw(cursor, cursorPosition, Color.White);

        spriteBatch.DrawString(spriteFont, text, textPosition, Color.Black);

        base.Draw(gameTime);
    }

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

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

Since this is a visual component, I had to derive it from **DrawableGameComponent** instead of **GameComponent**. I used two **Texture2D**'s for this component. One for the textbox and one for the cursor. I also needed variables for the **SpriteBatch** object, a **SpriteFont** and a **ContentManager**. The **ContentManager** wasn't exactly necessary but instead of passing in a lot of parameters to the constuctor I decided to load then in directly. There is also a variable for the text in the textbox.

The next variable might seem a little strange. To check to see if a key has been pressed, I created an array of all the keys that I wanted to check. That way, I could use a foreach loop in the **Update** method to go through the list of keys and see if one of the ones I'm interested in has been pressed. You will see this in action a little later on.

There are three **Vector2**'s in this component. One for the position of the textbox on the screen. There is another for the position of the text in the textbox. The last one is for the position of the cursor.

The purpose of the next two variables might be confusing. The one is a **TimeSpan** variable. I will use this variable to toggle the bool that will tell if the cursor is visible or not. It will effectively make the cursor blink.

Finally there are two more variables. They will be familiar to you. They will hold the current and the last states of the keyboard. They will be used in the familiar **CheckKey** method to see if a key has been pressed and released. I have explained before why I used this approach for detecting a single key press. If you do it the other way around and say the player has just hit the enter key to move from one menu to another. The key will still be in the same state when you move to the next menu. Doing it my way keeps this from happening.

The constructor for the component takes two parameters. The **Game** object and a **SpriteFont** for the component. Like most other components the constructor gets the **SpriteBatch** and **ContentManager** objects that were added to the list of services in the **Game1** class. The other things the constructor does are set the **SpriteFont**, load in the two textures that are needed for the textbox, initialize the **Vector2**'s set blink to false, which will make the cursor not visible to start, and the text to the empty string.

There are a few public properties for this component. There are get and set properties for the text of the textbox and the position of the textbox on the screen. In the set part for the property of the textbox, I call a method **SetTextPosition**. What this method does is set the position of the text in the textbox as well as the position of the cursor. In the new screen where I will need to position the textbox I needed to be able to get the height and the width of the textbox so there are get properties for that as well.

Next there is the **Update** method. The first thing the **Update** method does is get the current state of the keyboard. The next part may be new to you. This is the part where I control if the cursor is visible or not. The first thing I did was add the elapsed game time to the **TimeSpan** object **blinkTime**. If you are unfamiliar with that, basically what this does is get how much time has elapsed. In the if statement, I check to see if the time that has elapsed is greater then 500 milliseconds as half a second is a good

blink rate for the cursor. What happens if this is true is I negate the value of **blink**. Since **blink** is a boolean this will toggle it between true and false. Then I subtract 500 millisecond from the elapsed time, setting it back to it's previous value.

Then, I use a foreach loop to loop through all of the keys I'm interested in checking to see if it has been checked using the **CheckKey** method. If I find a key has been pressed I send it to a method called **AddKeyToText** and I exit the loop. After updating the text I get the length of the text and set the position of the cursor to the end of the text. When you are checking single keypresses the last thing you should do in the **Update** method is set the last state of the keyboard to the current state of the keyboard.

Next is the **AddKeyToText** method. It takes as a parameter the key press that was detected. In this method is a string variable that will hold the new text. If the length of the text is greater than or equal to 16, the maximum length of the text, and the key isn't the back space key, the key is ignored and the method returns. Then there is one huge switch that has each of the keys and what will be added to the text if the corrosponding key has been pressed. For example, if **Key.A** was pressed the new text is set to **a**. The interesting case is **Keys.Back**. What happens in this case is if there is text in the text for the text box it removes the last character. After the switch statement, if either the left or right shift key is pressed I call the **ToUpper** method to set the new character to an upper case character. Then I add the new text to the old text.

There is a **CheckKey** method in this class. It is the same as the other **CheckKey** methods and I explained it above so I will not go into any more details.

Next there is the **Draw** method. When the **Draw** method draws, in order, is the texture for the textbox, the cursor if it is visible and the text in the textbox. There is an if statement to see if the cursor is visible or not.

Like all of the other **DrawableGameComonpents** I've created, there are methods to show and hide the component. Since I didn't see the need to inherit from this one I didn't make them virtual so they could be overriden later on.

The next thing I did was create a new pop up screen called **InputScreen**. This screen will house the textbox control, a one button menu to allow the player to accept what they have typed as well as an image for the screen. Since this is a pop up screen it will be drawn in the center of the screen. This screen is basically the same as the **PopUpScreen** with only a few minor differences. (I actually just copied the **PopUpScreen** code and pasted it here to make this class.) This is the code for the **InputScreen**.

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

namespace New2DRPG
{
    class InputScreen : GameScreen
    {
        ButtonMenu menu;
```

```
Texture2D image;
SpriteBatch spriteBatch;
Rectangle imageRectangle;
Textbox textbox;

public InputScreen(Game game, SpriteFont spriteFont, Texture2D image,
          Texture2D buttonImage)
    : base(game)
{
    this.image = image;
    spriteBatch =
          (SpriteBatch)game.Services.GetService(typeof(SpriteBatch));
    imageRectangle = new Rectangle();
    imageRectangle.X = (game.Window.ClientBounds.Width - image.Width) / 2;
    imageRectangle.Y = (game.Window.ClientBounds.Height -
                                  image.Height) / 2;
    imageRectangle.Width = image.Width;
    imageRectangle.Height = image.Height;

    string[] items = { "OK" };
    menu = new ButtonMenu(game, spriteFont, buttonImage);
    menu.SetMenuItems(items);
    Components.Add(menu);
    textbox = new Textbox(game, spriteFont);
    Components.Add(textbox);
}

public int SelectedIndex
{
    get { return menu.SelectedIndex; }
}

public string Text
{
    get { return textbox.Text; }
}

public void SetMenuItems(string[] items)
{
    menu.SetMenuItems(items);
}

public override void Draw(GameTime gameTime)
{
    spriteBatch.Draw(image, imageRectangle, Color.White);
    base.Draw(gameTime);
}

public override void Show()
{
    base.Show();
    menu.Position = new Vector2((imageRectangle.Width -
                      menu.Width) / 2 + imageRectangle.X,
                      imageRectangle.Height - menu.Height - 10 +
                       imageRectangle.Y);
    textbox.Position = new Vector2((imageRectangle.Width -
                        textbox.Width) / 2 + imageRectangle.X,
                        menu.Position.Y - 10 - textbox.Height);
}
```

```
    }
}
```

As you can see, you need three using statements for this class. One for the XNA framework, another for the Graphics name space in the framework as well as the **CoreComponents** of the game. This class also inherits from **GameScreen**. There are variables in this class for the menu, the textbox, the **SpriteBatch** object, the image and the rectangle where the image will be drawn.

The constructor for this class takes the **Game** object, a **SpriteFont**, the texture for the image and the texture for the button. The constructor sets the image, gets the **SpriteBatch** object that was registered and then creates the rectangle where image will be drawn on the screen. It then creates a single entry button menu to allow the player to exit the screen and adds it to the list of components. It then creates a textbox and adds it to the list of components as well.

There are two get properties for this screen. There is one to get the selected menu item, which will always be 0, but later I do want to add in the ability to have a cancel button so I've kept this for now. There is also a property to get the text from the textbox. I also kept the method to be able to set the menu items for the screen. I thought it might come in handy later on.

The **Draw** method is exactly the same as the **Draw** method from the **PopUpScreen**. It just draws the image for the screen and then call **base.Draw**.

Another thing that is different from the **PopUpScreen** is the **Show** method. In the **Show** method as well as setting the position of the menu I also set the position of the textbox in the screen. I centered the textbox horizontally, using the **Width** property of the textbox. Then I positioned it above the menu using the height of the textbox and the same spacing I used for the menu.

To use this new screen, I did have to make some changes to the **CreatePCScreen**. I will give you the code for the entire class. I will go over the major changes. This is the new 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;

namespace New2DRPG
{
    class CreatePCScreen : GameScreen
    {
        ButtonMenu buttonMenu;

        string name = "Evander";
        bool gender = false;
        int difficultyLevel = 1;
        int className = 0;
        Texture2D buttonImage;
        int screenWidth;
        int screenHeight;
```

```
string[] classNames = {
          "Fighter",
          "Wizard",
          "Thief",
          "Priest" };

string[] menuItems = {
          "FORSEE HERO'S NAME",
          "FORSEE HERO'S GENDER",
          "FORSEE HERO'S CLASS",
          "FORSEE DIFFICULTY LEVEL",
          "BACK TO MENU",
          "BEGIN THE ADVENTURE" };

string[] difficultyLevels = {
            "Easy",
            "Normal",
            "Hard",
            "Ultimate" };

SpriteFont spriteFont;
SpriteBatch spriteBatch;
ContentManager Content;

public CreatePCScreen(Game game, SpriteFont spriteFont)
    : base(game)
{
    spriteBatch =
        (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
    Content =
        (ContentManager)Game.Services.GetService(typeof(ContentManager));
    this.spriteFont = spriteFont;
    Components.Add(new BackgroundComponent(game,
        Content.Load<Texture2D>("createpcscreen")));
    buttonImage = Content.Load<Texture2D>(@"GUI\buttonbackground");
    buttonMenu = new ButtonMenu(game, spriteFont, buttonImage);
    buttonMenu.SetMenuItems(menuItems);
    Components.Add(buttonMenu);

    screenWidth = Game.Window.ClientBounds.Width;
    screenHeight = Game.Window.ClientBounds.Height;
}

public bool Gender
{
    get { return gender; }
}

public int SelectedIndex
{
    get { return buttonMenu.SelectedIndex; }
}

public void ChangeName(string name)
{
    this.name = name;
}

public void ChangeGender(bool gender)
```

```
{
    this.gender = gender;
}

public void ChangeDifficulty(int difficultyLevel)
{
    this.difficultyLevel = difficultyLevel;
}

public void ChangeClass(int className)
{
    this.className = className;
}

public override void Show()
{
    buttonMenu.Position = new Vector2((screenWidth -
                      buttonMenu.Width) / 2,
                      screenHeight - buttonMenu.Height - 10);
    base.Show();
}

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[difficultyLevel];
    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);
}
```

```
    }
}
```

The first change is I got rid of the lists of female and male names. I made the name variable into a string instead of an int. I also gave the name a default value of Evander. It was just a name I came up with off the top of my head. The constructor didn't change at all. It is still the same. The **ChangeName** method takes a string instead of an int. The only other change was in the **Draw** method. I was able to get rid of the if statement and just use the name plus the " the " string.

To use the new classes I had to make a few changes to the **Game1** class. The first change was changing the screen for changing the name from **PopUpScreen** to **InputScreen**.

```
InputScreen nameInputScreen;
```

Then, I had to change the **LoadContent** method to reflect the changes made to this variable. All I did was change the name of the variable and how it was constructed. This is the updated **LoadContent** method.

```
        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>("titlescreen");
            startScreen = new StartScreen(this,
                normalFont,
                background,
                Content.Load<Texture2D>(@"GUI\buttonbackground"));
            Components.Add(startScreen);

            background = Content.Load<Texture2D>("helpscreen");
            helpScreen = new HelpScreen(this,
                normalFont,
                background,
                Content.Load<Texture2D>(@"GUI\buttonbackground"));
            Components.Add(helpScreen);

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

            background = Content.Load<Texture2D>(@"GUI\quitpopupbackground");
            quitPopUpScreen = new PopUpScreen(this,
                normalFont,
                background,
                Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
            Components.Add(quitPopUpScreen);
            quitPopUpScreen.Hide();

            background = Content.Load<Texture2D>(@"GUI\maleorfemale");

            genderPopUpScreen = new PopUpScreen(this,
```

```
                normalFont,
                background,
                Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
            string[] genderItems = new string[] { "Female", "Male" };
            genderPopUpScreen.SetMenuItems(genderItems);
            Components.Add(genderPopUpScreen);

            background = Content.Load<Texture2D>(@"GUI\chooseclass");

            classPopUpScreen = new PopUpScreen(this,
                normalFont,
                background,
                Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
            string[] classItems = new string[] { "Fighter", "Wizard", "Thief",
"Priest" };
            classPopUpScreen.SetMenuItems(classItems);
            Components.Add(classPopUpScreen);

            background = Content.Load<Texture2D>(@"GUI\choosedifficulty");

            difficultyPopUpScreen = new PopUpScreen(this,
                normalFont,
                background,
                Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
            string[] difficultyItems = new string[] { "Easy", "Normal", "Hard",
"Ultimate" };
            difficultyPopUpScreen.SetMenuItems(difficultyItems);
            Components.Add(difficultyPopUpScreen);

            background = Content.Load<Texture2D>(@"GUI\choosename");

            nameInputScreen = new InputScreen(this,
                normalFont,
                background,
                Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
            Components.Add(nameInputScreen);

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

            activeScreen = startScreen;
        }
```

I renamed the **HandleNamePopUpScreenInput** method to **HandleNameInputScreenInput**. Make sure you change it in both the **Update** method and the actual method. (If you change it in the actual method first, you can press SHIFT+ALT+F10 to change all occurances in your program.) This is the code for the **HandleNameInputScreenInput** method.

```
private void HandleNameInputScreenInput()
{
    if (CheckKey(Keys.Enter))
    {
        createPCScreen.ChangeName(nameInputScreen.Text);
        activeScreen.Hide();
        activeScreen = createPCScreen;
        activeScreen.Show();
    }
}
```

You will see that I removed checking for the space key to accept the screen. I will be doing that for all of the handle input methods. The only difference in this method is that instead of passing an integer to the **ChangeName** method, I pass in the string from the textbox.

Well, that is all for this tutorial. I will get to writing another one soon. Keep on coming back to the blog or the web site and I will try and have new stuff on both of them.