Eyes of the Dragon Tutorials
Part 13
List Box Control and Loading State

I'm writing these tutorials for the MonoGame 3.8 framework using Visual Studio 2019. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the Eyes of the Dragon page of my web blog. I will be making each version of the project available on GitHub here. It will be included on the page that links to the tutorials.

In going to be adding a **List Box** control in this tutorial and a loading state. I will be starting with the **List Box** control. One problem with adding a **List Box** is that I'm using the up and down arrow keys or the left thumb stick or direction pad on the control to navigate between controls. You would expect to navigate the items in a **List Box** the same way. The way that I'm going to handle this is having a control that can be selected activate the **List Box**. If the user presses the user presses enter or the A button to accept a selection control leaves, the same with pressing the escape key or B button. I am going to generate a basic texture for the control on the fly.

First, open the code for the **Control** class. I want to make the **HasFocus** property to be a virtual property. Change the **HasFocus** property to the following.

```
public virtual bool HasFocus
{
    get { return hasFocus; }
    set { hasFocus = value; }
}
```

Now, right click the **Controls** folder in the **MGRpgLibrary** project, select **Add** and then **Class**. Name this new class **ListBox**. This is the code for the **ListBox** class.

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

namespace MGRpgLibrary.Controls
{
    public class ListBox : Control
    {
        #region Event Region

        public event EventHandler SelectionChanged;
        public event EventHandler Enter;
        public event EventHandler Leave;

        #endregion

        #region Field Region

        List<string> items = new List<string>();
```

```
int startItem;
int lineCount;
Texture2D image;
Texture2D cursor;
Color selectedColor = Color.Red;
int selectedItem;

#endregion

#region Property Region

public Color SelectedColor
{
    get { return selectedColor; }
    set { selectedColor = value; }
}

public int SelectedIndex
{
    get { return selectedItem; }
    set { selectedItem = (int)MathHelper.Clamp(value, 0f, items.Count); }
}

public string SelectedItem
{
    get { return Items[selectedItem]; }
}

public List<string> Items
{
    get { return items; }
}

public override bool HasFocus
{
    get { return hasFocus; }

    set
    {
        hasFocus = value;

        if (hasFocus)
            OnEnter(null);
        else
            OnLeave(null);
    }
}

#endregion

#region Constructor Region

public ListBox(Texture2D background, Texture2D cursor)
    : base()
{
    hasFocus = false;
    tabStop = false;
    this.image = background;
    this.image = cursor;
    this.Size = new Vector2(image.Width, image.Height);
```

```
    lineCount = image.Height / SpriteFont.LineSpacing;
    startItem = 0;
    Color = Color.Black;
}

#endregion

#region Abstract Method Region

public override void Update(GameTime gameTime)
{
}

public override void Draw(SpriteBatch spriteBatch)
{
    spriteBatch.Draw(image, Position, Color.White);

    for (int i = 0; i < lineCount; i++)
    {
        if (startItem + i >= items.Count)
            break;

        if (startItem + i == selectedItem)
        {
            spriteBatch.DrawString(
                SpriteFont,
                items[startItem + i],
                new Vector2(Position.X, Position.Y + i * SpriteFont.LineSpacing),
                SelectedColor);
                spriteBatch.Draw(
                cursor,
                new Vector2(
                    Position.X - (cursor.Width + 2),
                    Position.Y + i * SpriteFont.LineSpacing + 5),
                    Color.White);
        }
        else
            spriteBatch.DrawString(
                SpriteFont,
                items[startItem + i],
                new Vector2(Position.X, 2 + Position.Y + i * SpriteFont.LineSpacing),
                Color);
    }
}
public override void HandleInput(PlayerIndex playerIndex)
{
    if (!HasFocus)
        return;

    if (InputHandler.KeyReleased(Keys.Down) ||
        InputHandler.ButtonReleased(Buttons.LeftThumbstickDown, playerIndex))
    {
        if (selectedItem < items.Count - 1)
        {

            selectedItem++;

            if (selectedItem >= startItem + lineCount)
                startItem = selectedItem - lineCount + 1;
```

```
                    OnSelectionChanged(null);
                }
            }
            else if (InputHandler.KeyReleased(Keys.Up) ||
                InputHandler.ButtonReleased(Buttons.LeftThumbstickUp, playerIndex))
            {
                if (selectedItem > 0)
                {
                    selectedItem--;

                    if (selectedItem < startItem)
                        startItem = selectedItem;

                    OnSelectionChanged(null);
                }
            }

            if (InputHandler.KeyReleased(Keys.Enter) ||
                InputHandler.ButtonReleased(Buttons.A, playerIndex))
            {
                HasFocus = false;
                OnSelected(null);
            }

            if (InputHandler.KeyReleased(Keys.Escape) ||
                InputHandler.ButtonReleased(Buttons.B, playerIndex))
            {
                HasFocus = false;
            }
        }

        #endregion

        #region Method Region

        protected virtual void OnSelectionChanged(EventArgs e)
        {
            if (SelectionChanged != null)
                SelectionChanged(this, e);
        }

        protected virtual void OnEnter(EventArgs e)
        {
            if (Enter != null)
                Enter(this, e);
        }

        protected virtual void OnLeave(EventArgs e)
        {
            if (Leave != null)
                Leave(this, e);
        }

        #endregion
    }
}
```

There are using statements for a few of the MonoGame framework name spaces like the other controls. The class inherits from **Control** so it can be added to the **ControlManager** class. I added three

events to the **List Box**. The first event, **SelectionChanged**, will be fired if user changes the selection by scrolling up or down. The **Enter** event is fired if the **ListBox** receives focus and the **Leave** event is fired if the **List Box** loses focus.

There are several new fields in the **List Box** class. Like the **Left Right Selector**, there is a **List<string>** for the items in the **ListBox**. There are then two integer fields **startItem** and **lineCount**. The **lineCount** field holds how many lines to draw in the **List Box**. The **startItem** field is where to start drawing items from, the top item in the **List Box**. The **image** field is a **Texture2D** for the **List Box**. The **cursor** field is also a **Texture2D** and will be drawn with the selected item. The **selectedColor** field is what color to draw the selected item in and the **selectedIndex** field is the index of the currently selected item.

There are properties to expose some of the new fields. The **SelectedColor** property is a read and write property and allows for changing the **selectedColor** field. The **SelectedIndex** property returns the **selectedIndex** field for the get part and in the set part clamps the value with in the range of items in the **List Box**. The **SelectedItem** property returns the item from the **List Box** at the **SelectedIndex**. The **Items** property returns the **items** field. The **HasFocus** property is a little more interesting than the others. The get part returns the **hasFocus** field of the parent class, **Control**. It is the set part that is more interesting. It sets the **hasFocus** field to the value passed in. Then I check to see what the value of **hasFocus** is. If it is true the **List Box** just received focus and I call the **OnEnter** method passing in **null** for the **EventArg**. Otherwise the **List Box** has lost focus and I call the **OnLeave** method again passing in **null** for the **EventArg**.

The constructor of the **List Box** class takes a **Texture2D** for the background image of the **List Box** and a **Texture2D** for the cursor. I set the **hasFocus** and **tabStop** fields to false. It is important not to have the **List Box** as a **tabStop** or you will break the **ControlManager** and the **List Box** won't work as expected. I set the **image** field to the **Texture2D** passed in as well as the **cursor** field. I then set the **Size** of the **List Box** to be the width and height of the **image**. To determine how many items will fit in the image I take the height of the image and divide that by the **LineSpacing** property of the **SpriteFont** of the **Control** base class. The **startItem** is set to 0, the first item, and the color for text to be drawn in is set to **Black**.

The **Draw** method first draws the **image** for the **ListBox**. There is then a loop that loops from zero to **lineCount**. There is an if statement inside the loop that checks to see if **startItem + i** is greater than or equal to the **Count** property of the **items**. If it is I break out of the loop. There is another if statement that compares **startItem + i** with the **selectedItem** field. If they are equal the string is drawn in using the **SelectedColor** property, otherwise it is drawn using the **Color** property. To determine where to draw the string I use the **X** value of the **ListBox**'s Position property. To find the **Y** value I take the **Y** value of the **List Box**'s Position property and add **i * SpriteFont.Linespacing**. Also, if the current item is the **selectedItem** I draw the **cursor** image to the left of the **List Box**. The **Y** position is the position of the item plus 5 pixels and the **X** position is the **X** position of the item minus the width of the cursor and 2 pixels.

The **HandleInput** method first checks to make sure the control has focus. If it doesn't I exit out of the method. I then check to see if the down key or the left thumb stick down has been released. If it is you want to scroll the selection down. I check to see if **selectedItem** is not at the end of the items. If it isn't

I increment **selectedItem** by 1. If **selectedItem** is greater or equal to the **startItem** plus the **lineCount** field then it is outside the items that fit into the **ListBox** and you want to move the **startItem** accordingly. **startItem** is set to be **selectedItem - LineCount + 1**. The selection has changed so I call **OnSelectionChanged** passing in **null** for the **EventArgs**.

For moving the selected item up I check to see if the up key or the left thumb stick up has been released. If **selectedItem** is not zero I decrease **selectedItem**. If **selectedItem** is less than **startItem** the selection is above the top item so I set **startItem** to be **selectedItem**. I then call **OnSelectionChanged** passing in **null** for the **EventArgs**.

There is then an if statement that checks to see if the enter key or the A button have been released. If they have I set the **HasFocus** property to **false**. It is important to use the property instead of the field. The property will fire the **Leave** event, if it is subscribed to. I then call the **OnSelected** method of the parent class passing in null for the **EventArgs**.

The last if statement checks to see if the escape key or the B button has been released. If they have I set the **HasFocus** property to **false**. Again, it is important to use the property rather than the field or the event won't fire.

There are then three methods: **OnSelectionChanged**, **OnEnter**, and **OnLeave**. These methods check to see if the event associated with them is subscribed to. If the event is subscribed to the event is fired passing in the **EventArgs** passed to the method.

I want to make a quick change to the **ControlManager** class. I want to add in a property, **AcceptInput**, that if set to false the **ControlManager** won't accept input. What I mean is that if you press the up or down key the selected control won't change. Add theses field and properties to the **ControlManager** class and change the **Update** method to the following. The **Update** method exits if the **AcceptInput** property is false.

```
bool acceptInput = true;

public bool AcceptInput
{
    get { return acceptInput; }
    set { acceptInput = value; }
}

        public void Update(GameTime gameTime, PlayerIndex playerIndex)
        {
            if (Count == 0)
                return;

            foreach (Control c in this)
            {
                if (c.Enabled)
                    c.Update(gameTime);

                if (c.HasFocus)
                    c.HandleInput(playerIndex);
            }
```

```
            if (!AcceptInput)
                return;

            if (InputHandler.ButtonPressed(Buttons.LeftThumbstickUp, playerIndex) ||
                InputHandler.ButtonPressed(Buttons.DPadUp, playerIndex) ||
                InputHandler.KeyPressed(Keys.Up))
                PreviousControl();

            if (InputHandler.ButtonPressed(Buttons.LeftThumbstickDown, playerIndex) ||
                InputHandler.ButtonPressed(Buttons.DPadDown, playerIndex) ||
                InputHandler.KeyPressed(Keys.Down))
                NextControl();
        }
```

I've been planning on adding a screen to load games, now is a good time to do that. Before I get to that screen though I want to change the **CharacterGeneratorScreen** and the **GamePlayScreen**. What I plan to do is to move the code for creating a character and the world from the **GamePlayScreen** into the **CharacterGeneratorScreen**. I'm going to make the **player** and **world** fields static and expose them using properties. Open to code for **GamePlayScreen** and change the code to the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using EyesOfTheDragon.Components;
using MGRpgLibrary;
using MGRpgLibrary.TileEngine;
using MGRpgLibrary.SpriteClasses;
using MGRpgLibrary.WorldClasses;

namespace EyesOfTheDragon.GameScreens
{
    public class GamePlayScreen : BaseGameState
    {
        #region Field Region

        Engine engine = new Engine(32, 32);
        static Player player;
        static World world;

        #endregion

        #region Property Region

        public static World World
        {
            get { return world; }
            set { world = value; }
        }

        public static Player Player
        {
            get { return player; }
            set { player = value; }
```

```
        }

        #endregion

        #region Constructor Region

        public GamePlayScreen(Game game, GameStateManager manager)
            : base(game, manager)
        {
        }

        #endregion

        #region XNA Method Region

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

        protected override void LoadContent()
        {
            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {

            world.Update(gameTime);
            player.Update(gameTime);

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            GameRef.SpriteBatch.Begin(
                SpriteSortMode.Deferred,
                BlendState.AlphaBlend,
                SamplerState.PointClamp,
                null,
                null,
                null,
                player.Camera.Transformation);
            base.Draw(gameTime);

            world.DrawLevel(GameRef.SpriteBatch, player.Camera);
            player.Draw(gameTime, GameRef.SpriteBatch);

            GameRef.SpriteBatch.End();
        }
        #endregion

        #region Abstract Method Region
        #endregion
    }
}
```

As you can see, it is much cleaner. The **Update** method just calls the **Update** methods of the **world** and

player fields. The **Draw** method just calls the **Draw** methods of the **world** and **player** fields as well. Now, open the code for the **CharacterGeneratorScreen**. First, make sure you have all of the using statements that you will need. Change the using statement area to the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MGRpgLibrary;
using MGRpgLibrary.Controls;
using MGRpgLibrary.SpriteClasses;
using MGRpgLibrary.TileEngine;
using MGRpgLibrary.WorldClasses;
using EyesOfTheDragon.Components;
```

What I did was in the **linkLabel1_Selected** method, after changing states, is call two methods. The first, **CreatePlayer**, creates a player object and the second, **CreateWorld**, creates a world object. Change the **linkLabel1_Selected** method to the following and add the **CreatePlayer** and **CreateWorld** methods to the **Method** region.

```
void LinkLabel1_Selected(object sender, EventArgs e)
{
    InputHandler.Flush();
    StateManager.ChangeState(GameRef.GamePlayScreen);
    CreatePlayer();
    CreateWorld();
}

private void CreatePlayer()
{
    Dictionary<AnimationKey, Animation> animations =
        new Dictionary<AnimationKey, Animation>();

    Animation animation = new Animation(3, 32, 32, 0, 0);
    animations.Add(AnimationKey.Down, animation);

    animation = new Animation(3, 32, 32, 0, 32);
    animations.Add(AnimationKey.Left, animation);

    animation = new Animation(3, 32, 32, 0, 64);
    animations.Add(AnimationKey.Right, animation);

    animation = new Animation(3, 32, 32, 0, 96);
    animations.Add(AnimationKey.Up, animation);

    int gender = genderSelector.SelectedIndex < 2 ? genderSelector.SelectedIndex : 1;

    AnimatedSprite sprite = new AnimatedSprite(
        characterImages[gender, classSelector.SelectedIndex],
        animations);

    GamePlayScreen.Player = new Player(GameRef, sprite);
}
```

```
private void CreateWorld()
{
    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");

    Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");

    Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    List<Tileset> tilesets = new List<Tileset>();

    tilesets.Add(tileset1);
    tilesets.Add(tileset2);

    MapLayer layer = new MapLayer(100, 100);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);

            layer.SetTile(x, y, tile);
        }
    }

    MapLayer splatter = new MapLayer(100, 100);

    Random random = new Random();

    for (int i = 0; i < 100; i++)
    {
        int x = random.Next(0, 100);
        int y = random.Next(0, 100);
        int index = random.Next(2, 14);

        Tile tile = new Tile(index, 0);

        splatter.SetTile(x, y, tile);
    }

    splatter.SetTile(1, 0, new Tile(0, 1));
    splatter.SetTile(2, 0, new Tile(2, 1));
    splatter.SetTile(3, 0, new Tile(0, 1));

    List<MapLayer> mapLayers = new List<MapLayer>();

    mapLayers.Add(layer);
    mapLayers.Add(splatter);

    TileMap map = new TileMap(tilesets, mapLayers);
    Level level = new Level(map);
    World world = new World(GameRef, GameRef.ScreenRectangle);

    world.Levels.Add(level);
    world.CurrentLevel = 0;

    GamePlayScreen.World = world;
}
```

The only code that you haven't seen before is where I create the **AnimatedSprite** object. I was able to use the **characterImages** array with the values of the **SelectedIndex** properties of **genderSelector** and **classSelected**. Before using the **genderSelector** I checked to see if it was less than two. If it was I assign that to a local variable **gender**. If it is greater than 1 I use 1.

Now it is time to add in the screen for loading games. Right click the **GameScreens** folder in the **EyesOfTheDragon** project, select **Add** and then **Class**. Name this class **LoadGameScreen**. This is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Content;
using MGRpgLibrary;
using MGRpgLibrary.Controls;
using MGRpgLibrary.SpriteClasses;
using MGRpgLibrary.TileEngine;
using MGRpgLibrary.WorldClasses;
using EyesOfTheDragon.Components;

namespace EyesOfTheDragon.GameScreens
{
    public class LoadGameScreen : BaseGameState
    {
        #region Field Region

        PictureBox backgroundImage;
        ListBox loadListBox;
        LinkLabel loadLinkLabel;
        LinkLabel exitLinkLabel;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region
        public LoadGameScreen(Game game, GameStateManager manager)
        : base(game, manager)
        {
        }

        #endregion

        #region Method Region

        protected override void LoadContent()
        {
            base.LoadContent();

            ContentManager Content = Game.Content;
                backgroundImage = new PictureBox(
                Content.Load<Texture2D>(@"Backgrounds\titlescreen"),
                    GameRef.ScreenRectangle);
```

```
            ControlManager.Add(backgroundImage);

            loadLinkLabel = new LinkLabel();
            loadLinkLabel.Text = "Select game";
            loadLinkLabel.Position = new Vector2(50, 100);
            loadLinkLabel.Selected += new EventHandler(loadLinkLabel_Selected);

            ControlManager.Add(loadLinkLabel);

            exitLinkLabel = new LinkLabel();
            exitLinkLabel.Text = "Back";
            exitLinkLabel.Position = new Vector2(50, 100 +
exitLinkLabel.SpriteFont.LineSpacing);
            exitLinkLabel.Selected += new EventHandler(exitLinkLabel_Selected);

            ControlManager.Add(exitLinkLabel);

            Texture2D texture = new Texture2D(GraphicsDevice, 300, 300);

            Color[] data = new Color[300 * 300];

            for (int i = 0; i < 300 * 300; i++)
            {
                data[i] = Color.White;
            }

            texture.SetData(data);

            loadListBox = new ListBox(
                texture,
                Content.Load<Texture2D>(@"GUI\rightarrowUp"));

            loadListBox.Position = new Vector2(400, 100);
            loadListBox.Selected += new EventHandler(loadListBox_Selected);
            loadListBox.Leave += new EventHandler(loadListBox_Leave);

            for (int i = 0; i < 20; i++)
                loadListBox.Items.Add("Game number: " + i.ToString());

            ControlManager.Add(loadListBox);
            ControlManager.NextControl();
        }

        public override void Update(GameTime gameTime)
        {
            ControlManager.Update(gameTime, PlayerIndex.One);

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            GameRef.SpriteBatch.Begin();

            base.Draw(gameTime);

            ControlManager.Draw(GameRef.SpriteBatch);
            GameRef.SpriteBatch.End();
        }
```

```
#endregion

#region Method Region

void loadListBox_Leave(object sender, EventArgs e)
{
    ControlManager.AcceptInput = true;
}

void loadLinkLabel_Selected(object sender, EventArgs e)
{
    ControlManager.AcceptInput = false;
    loadLinkLabel.HasFocus = false;
    loadListBox.HasFocus = true;
}

void loadListBox_Selected(object sender, EventArgs e)
{
    loadLinkLabel.HasFocus = true;
    loadListBox.HasFocus = false;

    ControlManager.AcceptInput = true;
    StateManager.ChangeState(GameRef.GamePlayScreen);

    CreatePlayer();
    CreateWorld();
}

void exitLinkLabel_Selected(object sender, EventArgs e)
{
    StateManager.PopState();
}

private void CreatePlayer()
{
    Dictionary<AnimationKey, Animation> animations = new Dictionary<AnimationKey,
        Animation>();

    Animation animation = new Animation(3, 32, 32, 0, 0);
    animations.Add(AnimationKey.Down, animation);

    animation = new Animation(3, 32, 32, 0, 32);
    animations.Add(AnimationKey.Left, animation);

    animation = new Animation(3, 32, 32, 0, 64);
    animations.Add(AnimationKey.Right, animation);

    animation = new Animation(3, 32, 32, 0, 96);
    animations.Add(AnimationKey.Up, animation);

    AnimatedSprite sprite = new AnimatedSprite(

        GameRef.Content.Load<Texture2D>(@"PlayerSprites\malefighter"),
        animations);

    GamePlayScreen.Player = new Player(GameRef, sprite);
}

private void CreateWorld()
```

```
        {
            Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
            Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

            tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");

            Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);
            List<Tileset> tilesets = new List<Tileset>();

            tilesets.Add(tileset1);
            tilesets.Add(tileset2);

            MapLayer layer = new MapLayer(100, 100);

            for (int y = 0; y < layer.Height; y++)
            {
                for (int x = 0; x < layer.Width; x++)
                {
                    Tile tile = new Tile(0, 0);
                    layer.SetTile(x, y, tile);
                }
            }

            MapLayer splatter = new MapLayer(100, 100);
            Random random = new Random();

            for (int i = 0; i < 100; i++)
            {
                int x = random.Next(0, 100);
                int y = random.Next(0, 100);
                int index = random.Next(2, 14);

                Tile tile = new Tile(index, 0);
                splatter.SetTile(x, y, tile);
            }

            splatter.SetTile(1, 0, new Tile(0, 1));
            splatter.SetTile(2, 0, new Tile(2, 1));
            splatter.SetTile(3, 0, new Tile(0, 1));

            List<MapLayer> mapLayers = new List<MapLayer>();

            mapLayers.Add(layer);
            mapLayers.Add(splatter);

            TileMap map = new TileMap(tilesets, mapLayers);
            Level level = new Level(map);
            World world = new World(GameRef, GameRef.ScreenRectangle);

            world.Levels.Add(level);
            world.CurrentLevel = 0;

            GamePlayScreen.World = world;
        }

        #endregion
    }
}
```

There are some using statements to bring some of the MonoGame Framework classes into scope as

well as classes from our **MGRpgLibrary**. There is also a using statement to bring the **Components** name space of **EyesOfTheDragon** into scope.

There is a **PictureBox** field for a background image, a **ListBox** field to hold the games that can be loaded, and two **LinkLabel** fields. The first, **loadLinkLabel**, will be used to activate the **ListBox** that holds the games. The second, **exitLinkLabel**, returns back to the start menu.

In the **LoadContent** method I create the controls on the form. The code for creating the **PictureBox** and **LinkLabels** is nothing new. Creating the **ListBox** will be new. First I generate a new textire for the list box. That is done by passing in the **GraphicsDevice** and the height and width of the texture. Once you have a texture you need to set its colour pixels. I do that by creating an array of **Color** the size of the height times the width. I then loop over all of the pixels and set their colour to white. The list box constructor takes two parameters: the image for the **ListBox** and an image for the cursor. I set the position of the **ListBox** to line up vertically with the **loadLinkLabel**. I then wire the **Selected** and **Leave** events of the **ListBox**. In a for loop I add 20 strings to the **ListBox**, to demonstrate how it works. I add it to the **ControlManager** and then call the **NextControl** method of the **ControlManager** class.

The **Update** method just calls the **Update** method of the **ControlManager** class. The **Draw** method wraps the **base.Draw** method call in calls to **Begin** and **End** of the **SpriteBatch** from the game and calls the **Draw** method of the **ControlManager** after the call to **base.Draw**.

The **LoadLinkLabel_Selected** method is where I handle if the **loadLinkLabel** has been selected. If it has I set the **AcceptInput** property of **ControlManager** to false. I then set the **HasFocus** property of **loadLinkLabel** to false and the **HasFocus** property of **loadListBox** to true. This essentially makes it so that the player can select items from the **List Box** with out the selected item in the control manager changing.

The **exitLinkLabel_Selected** method it where I handle if the **exitLinkLabel** has been selected. All that this method does is pop the current screen of the top of the state manager.

The **loadListBox_Selected** method is where I handle if the player hit the enter key when **loadListBox** was the active control. I set the **HasFocus** property of **loadLinkLabel** to true and the **AcceptInput** property of **ControlManager** to true so the selected control can be changed. I then change the state to the **GamePlayScreen** and call the **CreatePlayer** and **CreateWorld** methods.

The **loadListBox_Leave** method is where I handle if the player hit the escape key when **loadListBox** was the active control. I set the **HasFocus** method of **loadLinkLabel** to true and the **AcceptInput** property of **ControlManager** to true.
The **CreatePlayer** and **CreateWorld** methods create the player and world. The only difference is

where the **AnimatedSprite** is created for the player. For the **Texture2D** of the sprite I use the male fighter sprite. Eventually when you are able to write out and read in games you will load the sprite appropriate to the player's character.

I think that this is more than enough for this tutorial. I want to try and keep the tutorials to a reasonable length, unlike tutorial 11. So, I encourage you to visit my blog at,

https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck in your game programming adventures!
Cynthia

PS I may have some non-binary sprite so we don't have to fuss with the gender selector any more. I just need to check and see if the license allows for redistribution or not. Hopefully it does.