Eyes of the Dragon Tutorials 4.0

Part 11

Look Who's Talking, Now

I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. 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 4.0 page of my web blog. I will be making the project available on GitHub here. It will be included on the page that links to the tutorials.

This is part eleven of a series of tutorials I plan to write on creating a role-playing game with MonoGame. I've worked on similar tutorials using XNA and the past version of MonoGame. In the process of writing more tutorials, I discovered better ways of doing some things and had to go back to fix things. I'm hoping in this series to avoid making those same mistakes. Also, I am going to make the game cross-platform. In my previous tutorials, they were focused on Windows only. I want to open things up for macOS, Linux, Android, iOS, and any platform MonoGame supports.

This tutorial is in parallel to part 10 of A Summoner's Tale. We have a character on the map, and they are pretty but useless. They still don't actually do anything. It would be better if they could talk, or anything really. For that reason, we will continue adding conversations in this tutorial.

So, let's begin. First, I will add a class to RpgLibrary for serializing and deserializing content. In order to do that, we need to add a NuGet package to Psilibrary. Right-click Dependencies in the Psilibrary project and select Manage NuGet Packages. Click the Browse tab to browse NuGet packages and search for MonoGame.Framework.Content.Pipeline. Accept the licenses and add the package.

Now, I will add a class I used to serialize and deserialize content. Right-click the Psilibrary project, select Add and then Class. Name this new class XnaSerlializer. Yes, I've been using this class since my XNA days when the intermediate serializer was implemented in XNA 3.0, I believe. Here is the code for that class.

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

using Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate;
using System.Security.Cryptography;

namespace ConversationEditor
{
    public static class XnaSerializer
    {
        public static void Serialize<T>(string filename, T data)
        {
```

```
            XmlWriterSettings settings = new()
            {
                Indent = true
            };

            using XmlWriter writer = XmlWriter.Create(filename, settings);
            IntermediateSerializer.Serialize<T>(writer, data, null);
        }

        public static T Deserialize<T>(string filename)
        {
            T data = default;

            try
            {
                using FileStream stream = new(filename, FileMode.Open);
                using XmlReader reader = XmlReader.Create(stream);
                reader.Read();
                data = IntermediateSerializer.Deserialize<T>(reader, null);
            }
            catch (Exception)
            {

            }
            return data;
        }
    }
}
```

There are two generic methods: Serialize and Deserialize. The first is used to write content to XML, and the second is to read it. In the Serialize method, I create an XmlWriterSettings object that has the content indented, making it more readable. I then create an XmlWriter that the intermediate serializer will use to write the content. I then serialize the content. It is all very straightforward. However, it would be a good idea to wrap it in a try-catch block as I did with Deserialize.

In the Deserialize method, I assign a local variable to the default for the generic type. Then, in a try-catch block, I open a file stream wrapped in a using statement. Similarly, open an XmlReader passing in the stream. I then call the Read method to read the document. Finally, I use the Intermediate Serializer to deserialize the data and return the data.

I added a new type of character. I called them a villager. This character has a conversation attached to them. Right-click the SharedProject project, select Add and then New Folder. Name this new folder, Characters. Now, right-click the Characters folder, select Add and then Class. Name this new class, Villager. Here is the code for that class.

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework;
using RpgLibrary.Characters;
using RpgLibrary.TileEngine;
using SharedProject.Sprites;
using System;
using System.Collections.Generic;
using System.Text;

namespace SharedProject.Characters
```

```
{
    public class Villager : ICharacter
    {
        public string Name { get; set; }
        public AnimatedSprite AnimatedSprite { get; private set; }
        public Point Tile { get; set; }
        public Rectangle Bounds { get; private set; }
        public string Conversation { get; set; }
        public string Scene { get; set; }
        public bool Enabled { get; set; }
        public bool Visible { get; set; }
        public Vector2 Position { get; set; }

        public Villager(AnimatedSprite sprite, Point tile)
        {
            Enabled = true;
            Visible = true;
            Position = new();
            Tile = new();
            AnimatedSprite = sprite;
            Tile = tile;
            Bounds = new(Engine.PointToWorld(Tile), new(Engine.TileWidth, Engine.Tile-
Height));
        }

        public void Update(GameTime gameTime)
        {
            AnimatedSprite.Position = Position;
            AnimatedSprite.Update(gameTime);
        }

        public void Draw(SpriteBatch spriteBatch)
        {
            AnimatedSprite.Draw(spriteBatch);
        }
    }
}
```

The class implements the ICharacter interface. For properties, it has a name, animated sprite, tile, bound, conversation, scene, enabled, visible and position. The constructor takes an animated sprite and a tile for parameters. It initializes the properties. The Bounds property is calculated using the PointToWorld method on the Engine class. The Position property of the sprite is set to the Position property in the Update method. The Update method then calls the Update method of the sprite. Finally, the Draw method calls the Draw method of the sprite.

I added a new class, ConversationManager, that will manage all of our conversations. Add a new class, ConversationManager, to SharedProject. Here is the code for that class.

```
using ConversationEditor;
using Microsoft.VisualBasic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Psilibrary.ConversationComponents;
using Psilibrary.TileEngine;
using SummonersTale.SpriteClasses;
using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading.Tasks.Sources;

namespace SharedProject
{
    public interface IConversationManager
    {
        ConversationData GetConversation(string key);
        void AddConversation(string key, ConversationData conversation);
    }

    public class ConversationManager : GameComponent, IConversationManager
    {
        private readonly Dictionary<string, ConversationData> _conversations = new();
        private readonly Game _game;

        public Dictionary<string, ConversationData> Conversations => _conversations;

        public ConversationManager(Game game) : base(game)
        {
            game.Services.AddService<IConversationManager>(this);
        }

        public void AddConversation(string key, ConversationData conversationData)
        {
            if (!string.IsNullOrEmpty(key) && !Conversations.ContainsKey(key))
            {
                Conversations.Add(key, conversationData);
            }
        }

        public ConversationData GetConversation(string key)
        {
            if (Conversations.ContainsKey(key))
            {
                return Conversations[key];
            }

            return null;
        }

        public void LoadConverstions(ContentManager Content)
        {
            try
            {
                string folder = string.Format("{0}/Conversations/",
Content.RootDirectory);

                foreach (var f in Directory.GetDirectories(folder))
                {
                    string root = f.Replace(@"Content/", "");
                    string animation = root.Replace(string.Format(@"Conversations/"),
"");

                    foreach (var r in Directory.GetFiles(f))
                    {
                        string path = Path.GetFileNameWithoutExtension(r);
```

```
                        string build = string.Format(@"{0}/Conversations/{1}", root,
path);
                        ConversationData data = Content.Load<ConversationData>(build);
                        Conversations.Add(path, data);
                    }
                }
            }
            catch (Exception ex)
            {
            }
        }

        public void ReadConversations()
        {
            Conversations.Clear();

            try
            {
                foreach (var conversation in
Directory.GetFiles(string.Format("{0}/Conversations/", _game.Content.RootDirectory)))
                {
                    if (Path.GetExtension(conversation).ToLower() == "xml")
                    {

                        ConversationData data =
XnaSerializer.Deserialize<ConversationData>(conversation);
                        Conversations.Add(Path.GetFileNameWithoutExtension(conversation),
data);
                    }
                }
            }
            catch (Exception ex)
            {

            }
        }

        public void WriteConversations()
        {
            try
            {
                foreach (string conversation in Conversations.Keys)
                {
                    XnaSerializer.Serialize<ConversationData>(conversation,
Conversations[conversation]);
                }
            }
            catch (Exception ex)
            {

            }

        }

        public void LoadConverstions(Game game)
        {
            Conversation conversation = new();

            SceneAction action = new()
            {
```

```
                Action = ActionType.Talk,
                Parameter = "Help"
            };

            List<SceneOption> options = new()
            {
                new SceneOption("Help!", "Help", action)
            };

            action = new()
            {
                Action = ActionType.End,
                Parameter = ""
            };

            options.Add(new("Goodbye.", "Goodbye", action));

            GameScene scene = new(game, "Oh no! The unthinkable has happened! A thief has
stolen Greynar's eyes. Without them he will not be able to animate and defend us. You
have to do something or the monsters outside the village will crush us.", options);
            conversation.AddScene("Hello", scene);

            options = new()
            {
                new("Goodbye.", "Gooodbye", new() { Action = ActionType.End, Parameter =
"" })
            };

            scene = new(game, "Oh thank the heavens for you!", options);
            conversation.AddScene("Help", scene);

            conversation.FirstScene = "Hello";
            conversation.StartConversation();

            AddConversation("Rio", conversation);
        }
    }
}
```

There is an interface, IConversationManager, that will be used to register and retrieve the component. It has a method to add a new conversation to the manager a one to retrieve a conversation. The class inherits from GameComponent and implements the new interface.

There are two readonly fields. The first is a dictionary that contains our conversations. A second is a Game object. There is also a property that exposes the conversations. Finally, in the constructor, I registered the class as a service.

There are two checks before adding a conversation to the dictionary. The first is that the key is not null or empty and then that the key does not exist in the dictionary. If both are true, I add the conversation to the dictionary.

The GetConversation method accepts a string argument that is the name of the conversation to be retrieved. If the key is present in the dictionary, it is returned. Otherwise, null is returned.

The LoadConversations method is used to load the conversations. Instead of amalgamating all

conversations into one large file, I placed them in separate files. This method loads the conversations in from the disk. It requires a ContentManager in order to read them in. In a try-catch block, it creates a string that is the folder housing the conversations. Next, it loops over all of the folders in that folder.

Inside of that, I create a root folder for reading the content. I then loop over all of the files in the conversation folder. I get a string that is the name of the conversation on the disk using the GetFileNameWithoutExtension method on the file. I then create a string that is the string.Format method that points to the .xnb file. I read in the file using the ContentManager and then added it to the dictionary. So, it is important that your conversations have unique names, such as Hello1 and Hello2 rather than Hello and Hello. ReadConversations is virtually identical to LoadConversations. The difference is that ReadConversations use XnaSerializer instead of ContentManager to read the conversations.

WriteConversations is the reverse of LoadConversations and ReadConversations. In a try-catch block it loops over all of the conversation keys. It then calls the Serialize method of the XnaSerializer class, passing in the key and the conversation. I will circle back to this in a future tutorial.

There is a method LoadConversations that takes a Game parameter that was cut from the ConversationState in the last tutorial. It creates a test conversation programmatically like before and adds it to the dictionary.

We're moving along at a good pace. I made changes to the ConversationState, such as cutting out the code for creating conversations. Replace the ConversationState with this new version.

```
using Microsoft.Xna.Framework;
using Psilibrary.ConversationComponents;
using SummonersTale.Forms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RpgLibrary.ConversationComponents;
using SharedProject.Controls;
using SharpFont.Cache;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SharedProject.StateManagement
{
    public interface IConversationState
    {
        GameState GameState { get; }
        void SetConversation(Player player, string conversation);
        void StartConversation();
    }

    public class ConversationState : GameState, IConversationState
```

```
    {
        private IConversationManager _conversations;
        private Player player;
        private Conversation _conversation;
        public GameState GameState => this;
        public RenderTarget2D RenderTarget { get; set; }

        public ConversationState(Game game)
            : base(game)
        {
            Game.Services.AddService<IConversationState>(this);
            _conversations = Game.Services.GetService<IConversationManager>();
            _conversation = new();
        }

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

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

            RenderTarget = new(GraphicsDevice, Settings.TargetWidth, Settings.Tar-
getHeight);

            foreach (GameScene scene in _conversation.Scenes.Values)
            {
                scene.ItemSelected += Scene_ItemSelected;
            }
        }

        private void Scene_ItemSelected(object sender, SelectedIndexEventArgs e)
        {
            ButtonGroup btn = (ButtonGroup)sender;

            switch (btn.Action.Action)
            {
                case ActionType.End:
                    StateManager.PopState();
                    break;
                case ActionType.Talk:
                    _conversation.ChangeScene(btn.Action.Parameter);
                    break;
            }
        }

        public override void Update(GameTime gameTime)
        {
            _conversation.Update(gameTime);

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {

            GraphicsDevice.SetRenderTarget(RenderTarget);
            GraphicsDevice.Clear(Color.Transparent);
```

```
            SpriteBatch.Begin();

            base.Draw(gameTime);

            _conversation.Draw(gameTime, SpriteBatch);

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin();

            SpriteBatch.Draw(RenderTarget, new Rectangle(Point.Zero, Settings.Resolu-
tion), Color.White);

            SpriteBatch.End();
        }

        public void SetConversation(Player player, string conversation)
        {
            this.player = player;
            this._conversation = (Conversation)_conversations.GetConversation(conversa-
tion);
        }

        public void StartConversation()
        {
            _conversation.StartConversation();
        }
    }
}
```

The first change is I added SetConversation and StartConversation as members of the interface that must be implemented for IConversationState. There is a new field for the conversation manager and the conversation field was renamed to _conversation. In the constructor, I get the conversation manager from the list of services. In any other methods I use _conversation rather than conversation. So far, so good. Next stop, the gameplay state. First, we need a couple of new fields. One for the conversation manager and one from the conversation state. Add these two fields and property to the GamePlayState.

```
        public Player Player { get; private set; }

        private IConversationManager _conversationManager;
        private IConversationState _conversationState;
```

Next, we need to initialize them. The best place to do that is the LoadContent method. Replace that method with the following.

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

    Player = (Player)Game.Services.GetService<IPlayer>();
```

```
_conversationState = Game.Services.GetService<IConversationState>();
_conversationManager = Game.Services.GetService<IConversationManager>();

renderTarget = new(GraphicsDevice, Settings.BaseWidth, Settings.BaseHeight);

Texture2D texture = Game.Content.Load<Texture2D>(@"Tiles/tileset1");

List<Tileset> tilesets = new()
{
    new(texture, 8, 8, 32, 32),
};

TileLayer layer = new(100, 100);

map = new("test", tilesets[0], layer);

Dictionary<string, Animation> animations = new();

Animation animation = new(3, 32, 32, 0, 0) { CurrentFrame = 0, FramesPerSecond = 8 };
animations.Add("walkdown", animation);

animation = new(3, 32, 32, 0, 32) { CurrentFrame = 0, FramesPerSecond = 8 };
animations.Add("walkleft", animation);

animation = new(3, 32, 32, 0, 64) { CurrentFrame = 0, FramesPerSecond = 8 };
animations.Add("walkright", animation);

animation = new(3, 32, 32, 0, 96) { CurrentFrame = 0, FramesPerSecond = 8 };
animations.Add("walkup", animation);

texture = Game.Content.Load<Texture2D>(@"PlayerSprites/femalepriest");

sprite = new(texture, animations)
{
    CurrentAnimation = "walkdown",
    IsActive = true,
    IsAnimating = true,
};

texture = Game.Content.Load<Texture2D>(@"PlayerSprites/femalefighter");

AnimatedSprite rio = new(texture, animations)
{
    CurrentAnimation = "walkdown",
    IsAnimating = true,
};

CharacterLayer chars = new();

chars.Characters.Add(
    new Character("Rio", rio, "femalefighter")
    {
        Position = new(320, 320),
        Tile = new(10, 10),
        Visible= true,
        Enabled=true,
    });

map.AddLayer(chars);
```

```
    rightButton = new(Game.Content.Load<Texture2D>("GUI/g21245"), ButtonRole.Menu)
    {
        Position = new(80, Settings.BaseHeight - 80),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    rightButton.Down += RightButton_Down;
    ControlManager.Add(rightButton);

    upButton = new(Game.Content.Load<Texture2D>("GUI/g21263"), ButtonRole.Menu)
    {
        Position = new(48, Settings.BaseHeight - 48 - 64),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    upButton.Down += UpButton_Down;
    ControlManager.Add(upButton);

    downButton = new(Game.Content.Load<Texture2D>("GUI/g21272"), ButtonRole.Menu)
    {
        Position = new(48, Settings.BaseHeight - 48),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    downButton.Down += DownButton_Down;
    ControlManager.Add(downButton);

    leftButton = new(Game.Content.Load<Texture2D>("GUI/g22987"), ButtonRole.Menu)
    {
        Position = new(16, Settings.BaseHeight - 80),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    leftButton.Down += LeftButton_Down;

    ControlManager.Add(leftButton);
}
```

In addition to retrieving the instances of the ConversationManage and ConversationState, I now create a Villager instead of a Character. With the same values as before, other than setting the conversation.

Time to trigger a conversation. This will only be desktop, for now. I have some work to do on the mobile versions in another tutorial. What I did, is check to see if the F key has been released. If it has, I call a new method HandleConversation. Replace the Update method with this new version and add the HandleConversation method.

```
public override void Update(GameTime gameTime)
{
    Controls.Update(gameTime);
```

```
    sprite.Update(gameTime);
    Map.Update(gameTime);

    if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.A) && !inMotion)
    {
        MoveLeft();
    }
    else if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.D) && !inMotion)
    {
        MoveRight();
    }

    if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.W) && !inMotion)
    {
        MoveUp();
    }
    else if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.S) && !inMotion)
    {
        MoveDown();
    }

    if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.F) && !inMotion)
    {
        HandleConversation();
    }

    if (motion != Vector2.Zero)
    {
        motion.Normalize();
    }
    else
    {
        inMotion = false;
        return;
    }

    if (!sprite.LockToMap(new(99 * Engine.TileWidth, 99 * Engine.TileHeight), ref
motion))
    {
        inMotion = false;
        return;
    }

    Vector2 newPosition = sprite.Position + motion * speed *
(float)gameTime.ElapsedGameTime.TotalSeconds;

    Rectangle nextPotition = new Rectangle(
        (int)newPosition.X,
        (int)newPosition.Y,
        Engine.TileWidth,
        Engine.TileHeight);

    if (nextPotition.Intersects(collision))
    {
        inMotion = false;
        motion = Vector2.Zero;
        sprite.Position = new((int)sprite.Position.X, (int)sprite.Position.Y);
        return;
    }
```

```
    if (_tileMap.PlayerCollides(nextPotition))
    {
        inMotion = false;
        motion = Vector2.Zero;
        return;
    }

    sprite.Position = newPosition;
    sprite.Tile = Engine.VectorToCell(newPosition);

    _camera.LockToSprite(sprite, _tileMap);

    base.Update(gameTime);
}

private void HandleConversation()
{
    var layer = Map.Layers.FirstOrDefault(x => x is CharacterLayer);

    if (layer is CharacterLayer characterLayer)
    {
        foreach (ICharacter c in characterLayer.Characters)
        {
            if (c.Tile.X == sprite.Tile.X && Math.Abs(sprite.Tile.Y - c.Tile.Y) == 1 ||
                (c.Tile.Y == sprite.Tile.Y && Math.Abs(sprite.Tile.X - c.Tile.X) == 1))
            {
                if (c is Villager villager)
                {
                    _conversationState.SetConversation(Player, villager.Conversation);
                    manager.PushState((ConversationState)_conversationState);
                    //Conversation conversation =
(Conversation)_conversationManager.GetConversation(villager.Conversation);
                    //if (conversation != null)
                    //{
                    //}
                }
            }
        }
    }
}
```

The HandleConversation method uses a little LINQ to get the character layer. Since it is entirely possible null could be returned, there is an if statement that checks to see if what was returned is actually a character layer. Next, it loops over all of the characters on the layer. Now is the interesting part. I check to see if the X property of the player's sprite is the same as that of the character. If it is, I check to see if the Y property is one away. Similarly, it checks if the Y properties are the same and the X properties are within one. Unlike my other tutorial, I'm not forcing the player to be facing the sprite. That may happen in a future tutorial. If either of those conditions is true, I check to see if the character is a villager. If that is true, I set up the conversation.

That broke something. The map class does not have a property that exposes the layers. Add the following property to the TileMap class.

```
        public List<ILayer> Layers { get { return _mapLayers; } }
```

In the home stretch. The last thing to do is implement the conversation manager in the Desktop class and register a Player object. Since it is a relatively small class, I will give you the code for the entire class. Replace the Desktop class with the following version.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SharedProject;
using SharedProject.GameScreens;
using SharedProject.GamesScreens;
using SharedProject.StateManagement;

namespace EyesOfTheDragon
{
    public class Desktop : Game
    {
        private readonly GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        public GameStateManager GameStateManager { get; private set; }
        public ITitleState TitleState { get; private set; }
        public IStartMenuState StartMenuState { get; private set; }
        public IGamePlayState GamePlayState { get; private set; }
        public IConversationState ConversationState { get; private set; }

        public IConversationManager ConversationManager { get; private set; }
        public Player Player { get; private set; }

        public Desktop()
        {
            _graphics = new GraphicsDeviceManager(this);

            Content.RootDirectory = "Content";
            IsMouseVisible = true;

            _graphics.PreferredBackBufferWidth = Settings.BaseWidth;
            _graphics.PreferredBackBufferHeight = Settings.BaseHeight;
            _graphics.ApplyChanges();

            Components.Add(new Xin(this));

            GameStateManager = new GameStateManager(this);
            Components.Add(GameStateManager);
            Services.AddService(typeof(GameStateManager), GameStateManager);
        }

        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            Settings.TargetHeight = _graphics.PreferredBackBufferHeight;
            Settings.TargetWidth = _graphics.PreferredBackBufferWidth;

            base.Initialize();
        }

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

```
            Services.AddService(typeof(SpriteBatch), _spriteBatch);

            ConversationManager = new ConversationManager(this);
            Components.Add((ConversationManager)ConversationManager);
            ConversationManager.LoadConversations(this);

            TitleState = new TitleState(this);
            StartMenuState = new StartMenuState(this);
            GamePlayState = new GamePlayState(this);
            ConversationState = new ConversationState(this);

            GameStateManager.PushState((TitleState)TitleState);
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here

            base.Draw(gameTime);
        }
    }
}
```

The changes are that I added a new IConversationManager field for the conversation manager. Before creating the game states, I create an instance of ConversationManager. I call the LoadConversations passing in the instance of the game. I then call the WriteConversations method to write the conversations. Also, I move creating the states into the LoadContent method. You can build and run now. However, there will be one big error. When trying to render the conversation you will have a jet black screen. What happened is that it was rendering the render target to a rectangle with a height and width of zero. That is because I grabbed the whole copy of the conversation state from A Summoner's Tale, and it works slightly differently. Replace the Draw method of the ConversationState with this new version.

```
public override void Draw(GameTime gameTime)
{

    GraphicsDevice.SetRenderTarget(RenderTarget);
    GraphicsDevice.Clear(Color.Transparent);

    SpriteBatch.Begin();

    base.Draw(gameTime);

    _conversation.Draw(gameTime, SpriteBatch);
```

```
        SpriteBatch.End();

        GraphicsDevice.SetRenderTarget(null);

        SpriteBatch.Begin();

        SpriteBatch.Draw(RenderTarget, Settings.TargetRectangle, Color.White);

        SpriteBatch.End();
    }
```

There is just one last piece. I added another sprite font that will be used for rending the conversations. In the ShareProject, open the MGCB Editor. Right-click the Fonts folder and select New Item. Select Sprite Font Description from the list. Name this new font SceneFont. Leave it at the default settings for now.

If you build and run now, you can start a conversation with Rio. I'm going to park this tutorial here. There is more that I could pack in, but I will save it for the following tutorials. I don't want you to have too much to digest at once. I encourage you to visit the news page of my site, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials. Also, I'm thinking of reviving my newsletter so you will be informed of new stuff rather than having to keep looking for new things.

Good luck with your game programming adventures!
Cynthia