Eyes of the Dragon Tutorials
Part 35
Conversations

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.

So, the game is coming along well with NPCs on the map for the player to interact with. The player still cannot interact with them though. In this tutorial I will be covering conversations in depth. As a refresher a conversation is a collection of GameScene objects. A GameScene will render some text and give the player one or more SceneOptions to choose from. A SceneOption has a text property, scene property and SceneAction property. A SceneAction has an ActionType and a parameter.

Before I get into conversations I want to make two quick fixes to the GameScene class. First, I just want to change the location of where to load the scene background in the LoadContent method. Update that method to the following code.

```
protected void LoadContent(string textureName)
{
    texture = game.Content.Load<Texture2D>(@"GameScenes\" + textureName);
    selected = game.Content.Load<Texture2D>(@"GUI\rightarrowUp");
    font = game.Content.Load<SpriteFont>(@"Fonts\scenefont");
}
```

The other is in the constructor that takes a Game object, texture name, text and list of scene options. I missed setting the text field to the text passed in. I also updated how I assign the options. Instead of a straight assignment I add the options in a foreach loop. Update that constructor to the following.

```
public GameScene(Game game, string textureName, string text, List<SceneOption> options)
{
    this.game = game;
    this.textureName = textureName;
    this.text = text;

    LoadContent(textureName);

    this.options = new List<SceneOption>();

    foreach (var option in options)
    {
        this.options.Add(option);
    }

    highLight = Color.Red;
    normal = Color.Black;
}
```

Now, I'm going to walk over creating conversations using code. To do that I'm going to add a new method to the CharacterGeneratorScreen class and call it from CreateWorld. Update the CreateWorld

method to the following add add this using statement with the other using statements to bring the conversation classes into scope.

```
using MGRpgLibrary.ConversationComponents;

        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);
            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));

            TileMap map = new TileMap(tileset1, layer);

            map.AddTileset(tileset2);
            map.AddLayer(splatter);

            map.CollisionLayer.Collisions.Add(new Point(1, 0), CollisionType.Impassable);
            map.CollisionLayer.Collisions.Add(new Point(3, 0), CollisionType.Impassable);

            Level level = new Level(map);

            ChestData chestData = Game.Content.Load<ChestData>(@"Game\Chests\Plain Chest");
            Chest chest = new Chest(chestData);

            BaseSprite chestSprite = new BaseSprite(
                containers,
                new Rectangle(0, 0, 32, 32),
                new Point(10, 10));
```

```
            ItemSprite itemSprite = new ItemSprite(
                chest,
                chestSprite);

            level.Chests.Add(itemSprite);

            World world = new World(GameRef, GameRef.ScreenRectangle);

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

            AnimatedSprite s = new AnimatedSprite(
                GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
                AnimationManager.Instance.Animations);

            s.Position = new Vector2(5 * Engine.TileWidth, 5 * Engine.TileHeight);
            EntityData ed = new EntityData("Eliza", 10, 10, 10, 10, 10, 10, "20|CON|12", "16|
WIL|16", "0 | 0 | 0");
            Entity e = new Entity("Eliza", ed, EntityGender.Female, EntityType.NPC);
            NonPlayerCharacter npc = new NonPlayerCharacter(e, s);

            world.Levels[world.CurrentLevel].Characters.Add(npc);

            GamePlayScreen.World = world;

            CreateConversation();
        }

        private void CreateConversation()
        {

        }
```

I will now focus on the CreateConversation method. I will be doing it in blocks. You will add each block at the bottom of CreateConversation method. The first thing that we will need to do is create a conversation. That will be done with the constructor that takes the name of the conversation and the first scene. Add the following to the CreateConversation method.

```
Conversation c = new Conversation("eliza1", "welcome");
```

The convention I use for conversations in my games is to use the character's name and a number afterwards. This makes it clear what character the conversation belongs to and which conversation they are in.

The next thing that I'm going to do is create a GameScene object with no options. I will then generate the options and add them to the scene. Add the following line to the CreateConversation method.

```
GameScene scene = new GameScene(
    GameRef,
    "basic_scene",
    "The unthinkable has happened. A thief has stolen the eyes of the village guardian." +
    " With out his eyes the dragon will not animated if the village is attacked.",
    new List<SceneOption>());
```

For this scene I will be adding one option that when selected it will move to the next scene. Then I will add the scene to the conversation. Add the following code at the end of the method.

```
SceneAction action = new SceneAction
{
    Action = ActionType.Talk,
    Parameter = "none"
};

SceneOption option = new SceneOption("Continue", "welcome2", action);
scene.Options.Add(option);
c.AddScene("welcome", scene);
```

Since a scene option requires an action I create the action first. The action that will be taken is Action.Talk which moves the conversation to the node specified in the option. Action.Talk does not require a parameter so I set it to none. I create a scene option next that will display Continue. The next parameter is the scene that the conversation will transition to if the option is selected which is welcome2. The last parameter is the action that will be taken if the option is selected. The option is then added to the options for the scene and then the scene is added to the conversation.
Now I'm going to add a new scene with two options. Add the following code at the end of the CreateConversation method.

```
scene = new GameScene(
    GameRef,
    "basic_scene",
    "Will you retrieve the eyes of the dragon for us?",
    new List<SceneOption>());

action = new SceneAction
{
    Action = ActionType.Change,
    Parameter = "none"
};

option = new SceneOption("Yes", "eliza2", action);
scene.Options.Add(option);

action = new SceneAction
{
    Action = ActionType.Talk,
    Parameter = "none"
};

option = new SceneOption("No", "pleasehelp", action);
scene.Options.Add(option);

c.AddScene("welcome2", scene);
```

First, I create a new GameScene object using the existing variable. I then create a new action that will change the conversation to a different conversation using the ActionType.Change. This also does not require a parameter so I set that property to none. I now create the SceneOption with the values Yes for the text to be drawn, eliza2 for the conversation to change to and the action that was created. The SceneOption is then added to the collection of options. In a similar way I create a second option that will display No and move the current scene to pleasehelp. Finally I add the scene to the conversation.
There is one other scene that should be added to this conversation and that is the pleasehelp scene. Add the following code at the end of the CreateConversation method to add the last scene.

```
scene = new GameScene(
    GameRef,
```

```
    "basic_scene",
    "Please, you are the only one that can help us. If you change your mind " +
    "come back and see me.",
    new List<SceneOption>());

action = new SceneAction
{
    Action = ActionType.End,
    Parameter = "none"
};

option = new SceneOption("Bye", "welcome2", action);
scene.Options.Add(option);

c.AddScene("pleasehelp", scene);

ConversationManager.Instance.AddConversation("eliza1", c);
```

This follows the same flow as all of the other scenes. The difference is that the action type is End and the scene name passed in is scene that should be displayed when the player talks to the character the next time. I also add the conversation to the conversation manager because we are now finished with it.

Now I will add the second conversation to the conversation manager. Add the following code at the bottom of the CreateConversation method.

```
c = new Conversation("eliza2", "thankyou");

scene = new GameScene(
    GameRef,
    "basic_scene",
    "Thank you for agreeing to help us! Please find Faulke in the inn and ask " +
    "him what he knows about this thief.",
    new List<SceneOption>());

action = new SceneAction
{
    Action = ActionType.Quest,
    Parameter = "Faulke"
};

option = new SceneOption("Continue", "thankyou2", action);
scene.Options.Add(option);

c.AddScene("thankyou", scene);

scene = new GameScene(
    GameRef,
    "basic_scene",
    "Return to me once you've spoken with Faulke.",
    new List<SceneOption>());

action = new SceneAction
{
    Action = ActionType.End,
    Parameter = "none"
};
```

```
option = new SceneOption("Good Bye", "thankyou2", action);
scene.Options.Add(option);

c.AddScene("thankyou2", scene);

ConversationManager.Instance.AddConversation("eliza2", c);
```

The flow is pretty much the same as the other conversation that I added. I create a Conversation object names eliza2 with the first scene being thankyou. I create a new scene next. I then create an action. This action though is type ActionType.Quest. This action will give the player the quest with the name in the Parameter property of the object. I then add this option to the scene and the scene to the conversation. I then create a second scene. The action for this scene is ActionType.End and that ends the current conversation. The parameter for that is none as well. I then create an option and add the option to the scene. Then the scene is addedt to the conversation and the conversation is added to the conversation manager.

The last thing that needs to be done is attach the conversation to a character. First, I'm actually going to modify the NonPlayerCharacter class a bit. I'm going to remove the List<Converstation> and update the HasConversation property to reflect this as well as the constructor. I'm also going to add a method that set the currentConversation field to a conversation. Update the NonPlayerCharacter method to the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MGRpgLibrary.SpriteClasses;
using RpgLibrary.CharacterClasses;
using MGRpgLibrary.ConversationComponents;
using RpgLibrary.QuestClasses;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace MGRpgLibrary.CharacterClasses
{
    public class NonPlayerCharacter : Character
    {
        #region Field Region

        readonly List<Quest> quests;
        private string currentConversation;
        private string currentQuest;

        #endregion
        #region Property Region

        public List<Quest> Quests
        {
            get { return quests; }
        }

        public string CurrentConversation
        {
            get { return currentConversation; }
        }
```

```
        public string CurrentQuest
        {
            get { return currentQuest; }
        }

        public bool HasConversation
        {
            get { return !string.IsNullOrEmpty(currentConversation); }
        }

        public bool HasQuest
        {
            get { return (quests.Count > 0); }
        }

        #endregion

        #region Constructor Region

        public NonPlayerCharacter(Entity entity, AnimatedSprite sprite)
            : base(entity, sprite)
        {
            quests = new List<Quest>();
        }

        #endregion

        #region Method Region
        #endregion

        #region Virtual Method region

        public override void Update(GameTime gameTime)
        {
            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            base.Draw(gameTime, spriteBatch);
        }

        public void SetConversation(string conversation)
        {
            currentConversation = conversation;
        }

        #endregion
    }
}
```

Now, in the CreateWorld method of the CharacterGeneratorScreen replace the line the creates the NPC and places it on the map with the following two lines.

```
NonPlayerCharacter npc = new NonPlayerCharacter(e, s);
npc.SetConversation("eliza1");
```

All this does is assign a conversation to the NPC that we are creating and adding to the list of characters on the level. If you build and run the game and move the player close to the NPC you will

be able to start a conversation with the NPC. You will get the first scene that we added to the conversation. Other than that it does nothing. The one reason is that we only have one scene option for this conversation so even if you were to press one of the arrows the selected option would not change. The other is that there has been no logic added to the game to handle the player selecting a scene option.

The logic for changing handling actions will be done in the ConversationScreen class. Change the Update method of the ConversationScreen to the following.

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

    if (InputHandler.KeyReleased(Keys.Enter) || InputHandler.KeyReleased(Keys.Space) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One))
    {
        InputHandler.Flush();
        SceneAction action = conversation.CurrentScene.OptionAction;

        switch (action.Action)
        {
            case ActionType.Talk:
                conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                break;
            case ActionType.Quest:
                conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                break;
            case ActionType.Change:
                conversation =
                conversations.GetConversation(conversation.CurrentScene.OptionScene);
                conversation.StartConversation();
                break;
            case ActionType.End:
                StateManager.PopState();
                break;
        }
    }

    base.Update(gameTime);
}
```

What I did was add in an if statement that checks if the Enter key, Space key or the A button on the game pad has been released. If it has I call the Flush method to flush the input manager so that it doesn't move over into the next game frame. I then grab the action of the current game scene. Next up is a switch on the ActionType. This is where we will handle the individual acitons. For the Talk action I call the ChangeScene method of the conversation to change the scene to the OptionScene of the current scene. Currently I do the same thing for the Quest action. For the Change action I set the conversation using the GetConversation method of the conversation manager passing in the OptionScene. Next I call StartConversation to start that conversation. If the action is End I just pop the conversation state to return the game play state.

The next thing to do is make a couple of changes to the Draw method of the GameScene class. The first is that instead of checking that the texture is not null you should be checking if it is null. The other is that we are not drawing the texture. Change the Draw method of the GameScene class to the

following.

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Texture2D portrait)
{
    Vector2 selectedPosition = new Vector2();
    Rectangle portraitRect = new Rectangle(25, 25, 425, 425);
    Color myColor;

    if (selected == null)
        selected = game.Content.Load<Texture2D>(@"GUI\rightarrowUp");

    if (textPosition == Vector2.Zero)
        SetText(text);

    if (texture == null)
        texture = game.Content.Load<Texture2D>(@"GameScenes\" + textureName);

    spriteBatch.Draw(texture, Vector2.Zero, Color.White);

    if (portrait != null)
        spriteBatch.Draw(portrait, portraitRect, Color.White);

    spriteBatch.DrawString(font,
        text,
        textPosition,
        Color.White);

    Vector2 position = menuPosition;

    for (int i = 0; i < options.Count; i++)
    {
        if (i == SelectedIndex)
        {
            myColor = HighLightColor;
            selectedPosition.X = position.X - 35;
            selectedPosition.Y = position.Y;
            spriteBatch.Draw(selected, selectedPosition, Color.White);
        }
        else
            myColor = NormalColor;

        spriteBatch.DrawString(font,
            options[i].OptionText,
            position,
            myColor);

        position.Y += font.LineSpacing + 5;
    }
}
```

The last thing to do is add the background for the scenes. You can download my basic scene that I used from https://cynthiamcmahon.ca/blog/downloads/basic_scene.png. Once you have downloaded it you need to add it to the content project. Open the MonoGame Pipeline Tool. Right click the Content node in the tool, select Add and then New Folder. Name this new folder GameScenes. Right click the GameScenes folder, select Add and then Existing Item. Navigate to the basic-scene.png file you downloaded and add it. Save the project and close the MonoGame Pipeline Tool.

If you build and run now you will be able to start the conversation with the NPC that we added and the various options that were added will work. The one thing that will not happen is that the conversation changes are remembered. Also, the node that should start a quest currently does not start a quest. I'm not covering those in this tutorial though as the quest system needs to be updated a bit and for the conversation I need to add some plumbing that currently does exist.

I'm going to wrap up the tutorial here because I'd like to try and keep the tutorials to a reasonable length so that you don't have too much to digest at once. 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