Eyes of the Dragon Tutorials
Part 60
Quests

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, we have a lot of pieces in play now. Something that is missing, though, is quests. They are an integral part of a RPG. For that reason, the focus of this tutorial is going to be quests. First, I want to change the way the player interacts with NPCs. What I want to do is switch from using the space bar to using the F key. I know it is a minor thing, but I like that as part of the user experience. Replace the HandleConversation method of the GamePlayScreen to the following.

```
private void HandleConversation()
{
    if (InputHandler.KeyReleased(Keys.F) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One))
    {
        foreach (ILayer layer in World.Levels[World.CurrentLevel].Map.Layers)
        {
            if (layer is CharacterLayer)
            {
                foreach (Character c in ((CharacterLayer)layer).Characters.Values)
                {
                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        c.Sprite.Center);

                    if (distance < Character.SpeakingRadius && c is NonPlayerCharacter)
                    {
                        NonPlayerCharacter npc = (NonPlayerCharacter)c;

                        if (npc.HasConversation)
                        {
                            StateManager.PushState(GameRef.ConversationScreen);

                            GameRef.ConversationScreen.SetConversation(
                                player,
                                npc,
                                npc.CurrentConversation);

                            GameRef.ConversationScreen.StartConversation();
                        }
                    }
                    else if (distance < Character.SpeakingRadius && c is Merchant)
                    {
                        StateManager.PushState(GameRef.ShopScreen);
                        GameRef.ShopScreen.SetMerchant(c as Merchant);
                    }
                }
            }
        }
```

```
    }
}
```

Very minor tweak but something I've been meaning to do. Now, there are a few things to do in order to be ready for quests. First, I want to tweak the NonPlayerCharacter and Merchant classes. I want to draw an F above them if the player can interact with them. First, update the Merchant class' Draw method to the following.

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

    SpriteFont font = FontManager.GetFont("testfont");

    spriteBatch.DrawString(
        font,
        "F",
        new Vector2(
            Sprite.Bounds.X + (Sprite.Bounds.Width - font.MeasureString("F").X) / 2,
            Sprite.Bounds.Y - font.LineSpacing),
        Color.Yellow);
}
```

It explicitly calls the Draw method of the base class to draw the sprite. Then, it grabs the font from the font manager, to save from typing that over and over again. Next I draw an F above the merchant. I center it horizontally by taking the width of the sprite, subtracting the width of an F, dividing that by 2 and adding that to the X coordinate of the sprite's bounds. I position it above the sprite by taking the Y coordinate of the sprite's bounds and subtracting the LineHeight property of the font.

The NonPlayClass works essentially the same way. The difference is that instead of just drawing an F I check to see if the NPC has a quest or a conversation. Replace the Draw method of the NonPlayerCharacter class with the following.

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

    SpriteFont font = FontManager.GetFont("testfont");

    if (HasConversation || HasQuest)
    {
        spriteBatch.DrawString(
            font,
            "F",
            new Vector2(
                Sprite.Bounds.X + (Sprite.Bounds.Width - font.MeasureString("F").X) / 2,
                Sprite.Bounds.Y - font.LineSpacing),
            Color.Yellow);
    }
}
```

I also want to make a few changes to the Mob and Bandit classes. To the Mob class I want to add a property for the Name of the mob. It will be used in quests where you have to kill a certain number of mobs. Add the following property to the Mob class.

```
public string Name { get; protected set; }
```

So, in the Bandit class we need to set the Name property. Change the constructor of the Bandit class to the following.

```
public Bandit(Entity entity, AnimatedSprite sprite, Game game)
    : base(entity, sprite, game)
{
    _minGold = 20;
    _maxGold = 50;
    _xpValue = 100;

    Name = "Bandit";

    if (_game == null)
    {
        _game = game;
        LoadContent();
    }
}
```

Finally, I want to change the Draw method to draw the name over the bandit like I did with the NPC and merchant with the F. Update the Draw method of the Mob class as follows.

```
public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    sprite.Draw(gameTime, spriteBatch);

    SpriteFont font = FontManager.GetFont("testfont");

    Vector2 size = font.MeasureString(Name);
    Vector2 position = new Vector2(
        Sprite.Bounds.X + (Sprite.Bounds.Width - size.X) / 2,
        Sprite.Bounds.Y - font.LineSpacing);

    spriteBatch.DrawString(font, Name, position, Color.Red);
}
```

Now, on to quests. A quick recap. Quests are made up of a series of steps. Steps have different types, such as talk to a character or fight a foe. When you finish a quest you will get a reward. The reward will be gold, experience and possibly one or more items.

Before I get to quests and quest steps, I am going to add a data class that will be used to generate quest steps. In the MGRpgLibrary project right click on the Quests folder, select Add and then Class. Name this new class QuestStepData. Here is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace MGRpgLibrary.QuestClasses
{
    public enum QuestStepType { Talk, Buy, Sell, Deliver, Find, Fight }

    public class QuestStepData
```

```
    {
        public QuestStepType StepType { get; set; }
        public string Source { get; set; }
        public string Target { get; set; }
        public int Required { get; set; }
        public int Level { get; set; }
    }
}
```

I included an enumeration in this file for the type of quest step the data is for. For each type of quest step there will be a corresponding class. Then, each class will inherit from a base abstract class. The data class is made up of a series of public properties. There are properties for the type of step, the source of the step, the target of the step, how many times the step must be completed, and the level the player has to be to start the step.

Now, I am going to adjust the QuestStep class that we already had in the Quests folder. Replace that class with the following code.

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

namespace RpgLibrary.QuestClasses
{
    public abstract class QuestStep
    {
        #region Field Region
        #endregion

        #region Property Region

        public string Source { get; protected set; }
        public string Target { get; protected set; }
        public int Required { get; protected set; }
        public int Current { get; protected set; }
        public bool Finished { get; protected set; }
        public int Level { get; protected set; }
        public bool IsActive { get; protected set; }
        #endregion

        #region Constructor Region

        protected QuestStep()
        {
        }

        protected QuestStep(string source, string target, int required, int level)
        {
            Source = source;
            Target = target;
            Required = required;
            Finished = false;
            Level = level;
            IsActive = false;
            Current = 0;
        }
```

```
        #endregion

        #region Method Region
        #endregion

        #region Virtual Method region

        public abstract void Update(string target);

        public virtual void Start()
        {
            IsActive = true;
        }

        public virtual void End()
        {
            IsActive = false;
        }

        #endregion
    }
}
```

There are a series of properties first. They are the source, target, number of times the step needs to be repeated, the current number of times the step has been completed, if the step is finished, the level the player needs to be to start the quest, and if the step is currently active.

There are two protected constructors for the class. The first is a parameterless constructor that will be used in serialization. The second takes as parameters the source of the quest step, the target of the quest step, how many times it is required, and the level the player has to be to start the quest. It also initializes the properties to the values passed in or default values.

The is currently one abstract method: Update. This method takes a target. What I will be implementing first is a fight quest step. It will take the name of the mob to be defeated. In the case of a talk step, you will pass in the name of the speaker. Other steps will work in a similar manner. There is also a virtual method, Start, that starts a quest step. Finally, there is another virtual method, End, that ends a quest step. They are virtual in case they ever need to be overridden in the child classes.

I am going to add a child class now. This child is to fight mobs. Right click the QuestClasses folder in the Solution Explorer, select Add and then Class. Name this new class FightQuestStep. Here is the code for that class.

```
using RpgLibrary.QuestClasses;
using System;
using System.Collections.Generic;
using System.Text;

namespace MGRpgLibrary.QuestClasses
{
    public class FightQuestStep : QuestStep
    {
        private FightQuestStep() : base()
        {
```

```
        }

        private FightQuestStep(string source, string target, int required, int level)
            : base(source, target, required, level)
        {
        }

        public static QuestStep Create(QuestStepData data)
        {
            return new FightQuestStep(data.Source, data.Target, data.Required, data.Level);
        }

        public override void Update(string target)
        {
            if (Target.ToLower() == target.ToLower())
            {
                Current++;

                if (Current >= Required)
                {
                    Finished = true;
                }
            }
        }
    }
}
```

There are no new fields or properties in this class. It does have two private constructors that call the constructors of the base class with the parameters passed in. There is a static method, Create, that will be used to generate quest steps on the fly. It takes a QuestStepData parameter. It calls the second constructor passing in the properties of the data object passed to it. The Update method checks to see if the Target property of the object is the target pass in. If it is in increments the current by one. Then, if the current property is greater than or equal to the required property the step is marked as finished.

Before I get to the Quest class I want to add a class for rewards. As I mentioned earlier, a reward is an amount of gold, some experience, and possibly some items. Right click the QuestClasses folder in the Solution Explorer, select Add and then Class. Name this new class Reward. Here is the code for the Reward class.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace MGRpgLibrary.QuestClasses
{
    public class Reward
    {
        public int Experience { get; set; }
        public int Gold { get; set; }
        public List<string> Items { get; set; } = new List<string>();
    }
}
```

The first two properties are self-explanatory. They are the experience and gold gained for completing the quest. The third is a list of item names that the player will gain for completing the quest. When the

quest is completed the item will be pulled from the data managers.

At last we come to the Quest class. Replace the Quest class with the following code.

```
using MGRpgLibrary.QuestClasses;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RpgLibrary.QuestClasses
{
    public class Quest
    {
        #region Field Region

        public List<QuestStep> Steps { get; private set; } = new List<QuestStep>();
        public int CurrentStep { get; private set; }
        public string Source { get; private set; }
        public bool Finished { get; private set; }
        public Reward Reward { get; private set; }

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public Quest(string source, List<QuestStepData> steps, Reward reward)
        {
            Source = source;
            Reward = reward;

            foreach (var step in steps)
            {
                switch (step.StepType)
                {
                    case QuestStepType.Fight:
                        Steps.Add(FightQuestStep.Create(step));
                        break;
                }
            }
        }

        #endregion

        #region Method Region

        public bool Update(Character character, string target)
        {
            if (CurrentStep < Steps.Count)
            {
                Steps[CurrentStep].Update(target);

                if (Steps[CurrentStep].Finished)
                {
                    CurrentStep++;
                }
```

```
            }

            if (!Finished && CurrentStep >= Steps.Count)
            {
                Finished = true;
                character.Entity.AddExperience(Reward.Experience);
                character.UpdateGold(Reward.Gold);
            }

            return Finished;
        }

        #endregion

        #region Virtual Method region
        #endregion
    }
}
```

There is now a property that is the list of quest steps. There is a property that is the current step of the quest. There is a string that is the source of the quest. Also, there is a Reward property and a property for the state of the quest. The constructor takes for parameters the source of the quest, a List<QuestStepData>, and the reward. It initializes the properties. It iterates over the steps passed in and in a switch determines what kind of quest it is. Depending on the type of quest passed in it adds an object of that type to the list of QuestSteps.

There is also a method Update. It checks to see that the current step is within the bounds of the list. If it is, it calls the Update method of the current quest step. It then checks to see if the quest is complete. If the quest is complete, it rewards the character. I was going to do items in this class. However, the Backpack is in Player class and is not accessible in the library. I will handle that slightly differently. In the game when a quest completes I will add the items instead of handling it here.

So, the next step will be to add some quests to the game. I hate to say it, but that means altering the LoadWorld method of the CharacterGeneratorScreen. I will attach a quest to an NPC and assign its first QuestStep to active when the player picks a certain response in the conversation. Replace the LoadWorld method of the CharacterGeneratorScreen with the following.

```
private void LoadWorld()
{
    RpgLibrary.WorldClasses.LevelData levelData =
        Game.Content.Load<RpgLibrary.WorldClasses.LevelData>(@"Game\Levels\Starting Level");

    RpgLibrary.WorldClasses.MapData mapData =
        Game.Content.Load<RpgLibrary.WorldClasses.MapData>(@"Game\Levels\Maps\" +
levelData.MapName);

    CharacterLayerData charData =
        Game.Content.Load<CharacterLayerData>(@"Game\Levels\Chars\Starting Level");
    CharacterLayer characterLayer = new CharacterLayer();
    MobLayer mobLayer = new MobLayer();

    TileMap map = TileMap.FromMapData(mapData, Game.Content);

    foreach (var c in charData.Characters)
```

```
    {
        Character character;

        if (c.Value is NonPlayerCharacterData data)
        {
            Entity entity = new Entity(c.Value.Name, c.Value.EntityData, c.Value.Gender,
EntityType.NPC);

            using (Stream stream = new FileStream(c.Value.TextureName, FileMode.Open,
FileAccess.Read))
            {
                Texture2D texture = Texture2D.FromStream(GraphicsDevice, stream);
                AnimatedSprite sprite = new AnimatedSprite(texture,
AnimationManager.Instance.Animations)
                {
                    Position = new Vector2(c.Key.X * Engine.TileWidth, c.Key.Y *
Engine.TileHeight)
                };

                character = new NonPlayerCharacter(entity, sprite);

                ((NonPlayerCharacter)character).SetConversation(
                    data.CurrentConversation);
            }

            characterLayer.Characters.Add(c.Key, character);
        }
    }

    map.AddLayer(characterLayer);
    map.AddLayer(mobLayer);

    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)
    {
        Position = new Vector2(0 * Engine.TileWidth, 5 * Engine.TileHeight)
    };
```

```
    EntityData ed = new EntityData("Eliza", 1, 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);

    npc.SetConversation("brenda1");
    ((CharacterLayer)world.Levels[world.CurrentLevel].Map.Layers.Find(x => x is
CharacterLayer)).Characters.Add(new Point(0, 5), npc);
    world.Levels[world.CurrentLevel].Characters.Add(npc);

    s = new AnimatedSprite(
        GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
        AnimationManager.Instance.Animations)
    {
        Position = new Vector2(10 * Engine.TileWidth, 0)
    };

    ed = new EntityData("Barbra", 2, 10, 10, 10, 10, 10, 10, "20|CON|12", "16|WIL|16", "0|0|
0");

    e = new Entity("Barbra", ed, EntityGender.Female, EntityType.Merchant);

    Merchant m = new Merchant(e, s);

    Texture2D items = Game.Content.Load<Texture2D>("ObjectSprites/roguelikeitems");
    m.Backpack.AddItem(GameItemManager.GetItem("Long Sword"));
    m.Backpack.AddItem(GameItemManager.GetItem("Short Sword"));
    m.Backpack.AddItem(GameItemManager.GetItem("Apprentice Staff"));
    m.Backpack.AddItem(GameItemManager.GetItem("Acolyte Staff"));
    m.Backpack.AddItem(GameItemManager.GetItem("Leather Armor"));
    m.Backpack.AddItem(GameItemManager.GetItem("Chain Mail"));
    m.Backpack.AddItem(GameItemManager.GetItem("Studded Leather Armor"));
    m.Backpack.AddItem(GameItemManager.GetItem("Light Robes"));
    m.Backpack.AddItem(GameItemManager.GetItem("Medium Robes"));
    world.Levels[world.CurrentLevel].Characters.Add(m);
    ((CharacterLayer)world.Levels[world.CurrentLevel].Map.Layers.Find(x => x is
CharacterLayer)).Characters.Add(new Point(10, 0), m);
    GamePlayScreen.World = world;

    for (int i = 0; i < 25; i++)
    {
        ed = new EntityData("Bandit", 1, 10, 12, 12, 10, 10, 10, "20|CON|10", "12|WIL|12", "0|
0|0");

        e = new Entity("Bandit", ed, EntityGender.Male, EntityType.Monster);

        s = new AnimatedSprite(
            GameRef.Content.Load<Texture2D>(@"PlayerSprites/malerogue"),
            AnimationManager.Instance.Animations);

        Mob mob = new Bandit(e, s, GameRef);

        Rectangle r = new Rectangle(Mechanics.Random.Next(10, 50) * 32,
Mechanics.Random.Next(10, 50) * 32, 32, 32);

        mob.Sprite.Position = new Vector2(r.X, r.Y);
```

```
        if (!mobLayer.Mobs.ContainsKey(r))
        {
            mobLayer.Mobs.Add(r, mob);
        }

        mob.Entity.Equip(GameItemManager.GetItem("Short Sword"));
        mob.Drops.Add(GameItemManager.GetItem("Short Sword"));
        mob.Drops.Add(GameItemManager.GetItem("Minor Healing Potion"));
    }

    QuestStepData step = new QuestStepData()
    {
        StepType = QuestStepType.Fight,
        Level = 1,
        Source = "Eliza",
        Target = "Bandit"
    };

    List<QuestStepData> steps = new List<QuestStepData>
    {
        step
    };

    Reward reward = new Reward { Experience = 1000, Gold = 1000 };
    reward.Items.Add("Minor Healing Potion");

    Quest q = new Quest("Eliza", steps, reward);

    npc.Quests.Add(q);
}
```

I created a new NPC and positioned her at (0, 5) just before I created the merchant. After creating the mobs I create a new QuestStepData object. I will attach it to the new NPC. It is unlocked at level 1, it is a fight quest step and the target will be bandits. Following that is a List of QuestStepData initialized with the step I just created. I then create a reward of 1000 gold and 100 experience. To that I also attach a minor healing potion. Next I create a quest, then attach it to the npc I created earlier.

We also need to create the conversation. The first step will be to update the LinkLabel1_Selected method to call the CreateConversation method. Replace the LinkLabel1_Selected method with the following code.

```
void LinkLabel1_Selected(object sender, EventArgs e)
{
    InputHandler.Flush();

    CreatePlayer();
    LoadWorld();
    CreateConversation();

    GameRef.SkillScreen.SkillPoints = 10;

    Transition(ChangeType.Change, GameRef.SkillScreen);

    GameRef.SkillScreen.SetTarget(GamePlayScreen.Player.Character);
}
```

The next step will be to create the conversation in the CreateConversation method. Replace that method with the following code.

```
private void CreateConversation()
{
    ConversationManager.Instance.ConversationList.Clear();
    Conversation c = new Conversation("eliza1", "welcome");
    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>());

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

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

    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.Quest,
        Parameter = "none"
    };

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

    c.AddScene("welcome2", 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);

    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);

    c = new Conversation("brenda1", "bandits");
    scene = new GameScene(
        GameRef,
        "basic_scene",
        "With the eyes gone. The bandits are getting bolder. Will you please help thin their
numbers." +
        " Even killing two of them would be helpful.",
        new List<SceneOption>());

    action = new SceneAction
    {
```

```
    Action = ActionType.Talk,
    Parameter = "none"
};

option = new SceneOption("Continue", "bandits2", action);
scene.Options.Add(option);
c.AddScene("bandits", scene);

scene = new GameScene(
    GameRef,
    "basic_scene",
    "Will you help with the bandits?",
    new List<SceneOption>());

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

option = new SceneOption("Yes", "brenda2", 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("bandits2", 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("brenda1", c);

c = new Conversation("brenda2", "thankyou");

scene = new GameScene(
    GameRef,
    "basic_scene",
    "Thank you for agreeing to help us! The bandits are south of the village.",
```

```
            new List<SceneOption>());

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

        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 dealt with the bandits.",
            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("brenda2", c);
    }
```

Since we have created conversations before, I will just gloss over it. First, you create a conversation. Then you create a scene. Once you have a scene you create options and attach them to the scene. Once the scene is created you add it to the conversation. The conversation is then added to conversation manager. I create a scene that is an introduction. It then transitions to a choice between yes helping, which has an action type of Quest, and now which just moves to the next conversation. I then create the other conversations.

Before we can attack a quest to the player we need to add a property that will hold their active quests. Add the following property to the Player class.

```
public List<Quest> Quests { get; internal set; } = new List<Quest>();
```

Now, I need to attach the quest to the player. However, we currently do not have an option to give the player a quest. To do that we need to modify the ConversationScreen class. Update it as follows.

```
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.CharacterClasses;
using MGRpgLibrary.ConversationComponents;

namespace EyesOfTheDragon.GameScreens
{
    public class ConversationScreen : BaseGameState
    {
        private ConversationManager conversations = ConversationManager.Instance;
        private Conversation conversation;
        private Player player;
        private NonPlayerCharacter npc;

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

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

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

        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.CheckMouseReleased(MouseButton.Left) &&
conversation.CurrentScene.IsOver))
            {
                InputHandler.Flush();
                SceneAction action = conversation.CurrentScene.OptionAction;

                switch (action.Action)
                {
                    case ActionType.Talk:
                        conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                        break;
                    case ActionType.Quest:
                        if (npc.HasQuest && npc.Quests[0].Steps[0].Level <=
player.Character.Entity.Level)
                        {
                            npc.Quests[0].Steps[0].Start();
                            GamePlayScreen.Player.Quests.Add(npc.Quests[0]);
                        }
                        conversation =
                        conversations.GetConversation(conversation.CurrentScene.OptionScene);
                        conversation.StartConversation();
                        break;
                    case ActionType.Change:
                        conversation =
                        conversations.GetConversation(conversation.CurrentScene.OptionScene);
                        conversation.StartConversation();
                        break;
```

```
                    case ActionType.End:
                        StateManager.PopState();
                        break;
                }
            }

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);

            GameRef.SpriteBatch.Begin();

            conversation.Draw(gameTime, GameRef.SpriteBatch, null);

            GameRef.SpriteBatch.End();
        }

        public void SetConversation(Player player, NonPlayerCharacter npc, string
            conversation)
        {
            this.player = player;
            this.npc = npc;
            this.conversation = conversations.GetConversation(conversation);
        }

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

I really only updated the Update method. I updated the case for the Quest action type. What I do is check to see if the level of the quest is less than or equal to the level of the player's character. If it is I call the Start method on the first step of the quest. Also, I add the quest to the list of active quests for the player. I then get the conversation to change to. Finally, I start that conversation.

We are getting there. There are just a few steps left. The next step is to update the quest. That is done in the HandleMobs method of the GamePlayScreen, since we only have a FightQuestStep currently. Change the HandleMobs method to the following code.

```
private void HandleMobs(GameTime gameTime)
{
    MobLayer mobLayer = World.Levels[World.CurrentLevel].Map.Layers.Find(x => x is MobLayer) as
MobLayer;

    if (playerAttacking)
    {
        foreach (var mob in mobLayer.Mobs.Values)
        {
            if (playerSword.Intersects(mob.Sprite.Bounds))
            {
                if (player.Character.Entity.MainHand != null &&
                    player.Character.Entity.MainHand.Item is Weapon)
```

```
                {
                    mob.Entity.ApplyDamage(player.Character.Entity.MainHand);
                    playerAttacking = false;

                    if (mob.Entity.Health.CurrentValue <= 0)
                    {
                        StateManager.PushState(GameRef.LootScreen);
                        GamePlayScreen.Player.Character.Entity.AddExperience(mob.XPValue);
                        GameRef.LootScreen.Gold = mob.GoldDrop;

                        foreach (var i in mob.Drops)
                        {
                            GameRef.LootScreen.Items.Add(i);
                        }
                    }
                }
            }
        }
    }

    foreach (var mob in mobLayer.Mobs.Where(kv => kv.Value.Entity.Health.CurrentValue <=
0).ToList())
    {
        foreach (Quest q in Player.Quests)
        {
            foreach (QuestStep s in q.Steps)
            {
                s.Update(mob.Value.Entity.EntityName);
            }
        }

        mobLayer.Mobs.Remove(mob.Key);
    }

    foreach (var mob in mobLayer.Mobs.Values)
    {
        mob.DoAttack(player.Sprite, player.Character.Entity);
        mob.ShouldAttack(player.Sprite);
    }

    foreach (var mob in mobLayer.Mobs.Values)
    {
        float distance = Vector2.Distance(player.Sprite.Center, mob.Sprite.Center);

        if (distance < mob.Sprite.Width * 4)
        {
            Vector2 motion = Vector2.Zero;

            if (mob.Sprite.Position.X < player.Sprite.Position.X)
            {
                motion.X = 1;
                mob.Sprite.CurrentAnimation = AnimationKey.Right;
            }

            if (mob.Sprite.Position.X > player.Sprite.Position.X)
            {
                motion.X = -1;
                mob.Sprite.CurrentAnimation = AnimationKey.Left;
            }
```

```
            if (mob.Sprite.Position.Y < player.Sprite.Position.Y)
            {
                motion.Y = 1;
                mob.Sprite.CurrentAnimation = AnimationKey.Down;
            }

            if (mob.Sprite.Position.Y > player.Sprite.Position.Y)
            {
                motion.Y = -1;
                mob.Sprite.CurrentAnimation = AnimationKey.Up;
            }

            if (motion != Vector2.Zero)
            {
                motion.Normalize();
            }

            float speed = 200f;

            motion *= speed * (float)gameTime.ElapsedGameTime.TotalSeconds;

            mob.Sprite.Position += motion;
            mob.Sprite.IsAnimating = true;

            if (mob.Sprite.Bounds.Intersects(player.Sprite.Bounds))
            {
                mob.Sprite.Position -= motion;
            }
        }
        else
        {
            mob.Sprite.IsAnimating = false;
        }
    }
}
```

When we check to remove a mob from the game there are nested foreach loops. The outer loop will loop over all of the quests the player has. The inner loop loops over the steps and calls the Update method passing in the name of the mob.

The only thing left is to turn in the quest and remove the quest from the NPC's quest list. Because we are working with reference types, that will also be done in the GamePlayScreen. Replace the method with the following code.

```
private void HandleConversation()
{
    if (InputHandler.KeyReleased(Keys.F) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One))
    {
        foreach (ILayer layer in World.Levels[World.CurrentLevel].Map.Layers)
        {
            if (layer is CharacterLayer layer1)
            {
                foreach (Character c in layer1.Characters.Values)
                {
                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        c.Sprite.Center);
```

```
                    if (distance < Character.SpeakingRadius && c is NonPlayerCharacter
character)
                    {
                        NonPlayerCharacter npc = character;

                        if (npc.Quests.Count > 0 && npc.Quests[0].Finished)
                        {
                            Reward r = npc.Quests[0].Reward;

                            player.Gold += r.Gold;
                            player.Character.Entity.AddExperience(r.Experience);

                            npc.Quests.RemoveAt(0);
                        }

                        if (npc.HasConversation)
                        {
                            StateManager.PushState(GameRef.ConversationScreen);

                            GameRef.ConversationScreen.SetConversation(
                                player,
                                npc,
                                npc.CurrentConversation);

                            GameRef.ConversationScreen.StartConversation();
                        }
                    }
                    else if (distance < Character.SpeakingRadius && c is Merchant)
                    {
                        StateManager.PushState(GameRef.ShopScreen);
                        GameRef.ShopScreen.SetMerchant(c as Merchant);
                    }
                }
            }
        }
    }
}
```

What I do is check to see if the first quest that the NPC has is finished. As I mentioned earlier, because we are updating reference types when we update the player's quest we are also updating the NPC's quest. So, when the player's quest finishes, so does the NPCs. Also, I need to check to make sure there is a quest. I then grab the reward for the quest. I update the player's gold and experience. Currently, I do not handle the drops. That is going to add another layer of complexity that I don't want to get into in this tutorial. Finally, I remove the quest.

If you build and run now you, you will get an error processing the map. That is because there are now quests but, we didn't add them to the editor. The easiest solution is to modify the XML so that the quests are null. Update the StartingLevel.xml file in the folder Content\Game\Levels\Chars to the following.

```
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:TileEngine="MGRpgLibrary.TileEngine"
xmlns:CharacterClasses="MGRpgLibrary.CharacterClasses">
  <Asset Type="TileEngine:CharacterLayerData">
    <Characters>
```

```
      <Item>
        <Key>11 9</Key>
        <Value Type="CharacterClasses:NonPlayerCharacterData">
          <Name>Eliza</Name>
          <Gender>Female</Gender>
          <EntityData>
            <EntityName>Eliza</EntityName>
            <Level>1</Level>
            <Strength>10</Strength>
            <Dexterity>10</Dexterity>
            <Cunning>10</Cunning>
            <Willpower>10</Willpower>
            <Magic>10</Magic>
            <Constitution>10</Constitution>
            <HealthFormula></HealthFormula>
            <StaminaFormula></StaminaFormula>
            <MagicFormula></MagicFormula>
          </EntityData>
          <TextureName>C:\Users\cynth\Documents\GitHub\MG-
EyesOfTheDragon\EyesOfTheDragon\EyesOfTheDragon\Content\SpriteSheets\Eliza.png</TextureName>
          <Head>
            <Position>0 0</Position>
            <TextureName Null="true" />
            <SourceRectangle Null="true" />
            <BaseItem Null="true" />
            <Type Null="true" />
          </Head>
          <Torso>
            <Position>0 0</Position>
            <TextureName Null="true" />
            <SourceRectangle Null="true" />
            <BaseItem Null="true" />
            <Type Null="true" />
          </Torso>
          <Hands>
            <Position>0 0</Position>
            <TextureName Null="true" />
            <SourceRectangle Null="true" />
            <BaseItem Null="true" />
            <Type Null="true" />
          </Hands>
          <Feet>
            <Position>0 0</Position>
            <TextureName Null="true" />
            <SourceRectangle Null="true" />
            <BaseItem Null="true" />
            <Type Null="true" />
          </Feet>
          <MainHand>
            <Position>0 0</Position>
            <TextureName Null="true" />
            <SourceRectangle Null="true" />
            <BaseItem Null="true" />
            <Type Null="true" />
          </MainHand>
          <OffHand>
            <Position>0 0</Position>
            <TextureName Null="true" />
            <SourceRectangle Null="true" />
            <BaseItem Null="true" />
```

```
            <Type Null="true" />
          </OffHand>
          <Quests Null="true" />
          <CurrentConversation Null="true" />
          <CurrentQuest Null="true" />
        </Value>
      </Item>
    </Characters>
  </Asset>
</XnaContent>
```

If you build and run now, you can talk to the NPC at (0, 5) and get their quest. You can also finish it and turn it in. There are a few problems that I will address in the next tutorial. The first problem is that the NPC's conversation keeps reverting to the base conversation. Second, there is no item drops. As I mentioned, that is a layer of complexity that I don't want to get into in this tutorial.

That is it for this tutorial. I accomplished what I had planned on, and I don't want to get into something new at this point. I accomplished most of what I intended, and I don't want to venture further in this tutorial. So, please continue to visit my blog, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck with your Game Programming Adventures!

Cynthia