Eyes of the Dragon Tutorials
Part 61
Quests – Continued

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, in the last tutorial we did a lot of work on quests. There were somethings missing, though. In this tutorial I would like to finish quests. In the last tutorial we were working on quest rewards. We have quests being completed, and it is possible to turn in quests. What wasn't working was item rewards. I forgot that we have a GameItemManager class to manage game items. I thought I was going to have to create one. Since we have it, I will make use of it. Ideally you would want to use an editor to create a file that could be read in at runtime to create the source rectangles for items. However, I am not going to do that in this tutorial. I will just make do with what he in place already.

First, I'm sorry. I didn't fully test the game before publishing the tutorial. You could complete a quest step, but you could not complete a quest. To remedy that, I added a method to the Quest class that would check if a quest was complete or not. Add the following method to the Quest class.

```
public void CheckFinished()
{
    Finished = true;

    foreach (var s in Steps)
    {
        if (!s.Finished)
        {
            Finished = false;
            break;
        }
    }
}
```

This method sets the Finished property to true. It then loops over all of the steps in the quest. If there is a step that is not finished it sets Finished to false and breaks out of the loop. There was another bug that I found while testing, but that will have to wait. First, I want to add a new state that displays the player's reward for completing the quest. Right click the GameScreens folder in the EyesOfTheDragon project, select Add and then Class. Name this new class QuestCompleteState, I know, it is a long name. Here is the code for that class.

```
using MGRpgLibrary;
using MGRpgLibrary.Controls;
using MGRpgLibrary.ItemClasses;
using MGRpgLibrary.QuestClasses;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
```

```
using System.Text;

namespace EyesOfTheDragon.GameScreens
{
    public class QuestCompleteState : BaseGameState
    {
        private Reward _reward;
        private Label _gold;
        private Label _exp;

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

        public void SetReward(Reward reward)
        {
            _reward = reward;
        }

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

            Vector2 size = FontManager.GetFont("testfont").MeasureString("Close");

            LinkLabel label = new LinkLabel()
            {
                Position = new Vector2(
                    (Game1.ScreenWidth - size.X) / 2,
                    720 - FontManager.GetFont("testfont").LineSpacing * 2),
                Text = "Close",
                Visible = true
            };

            ControlManager.Add(label);

            Label l = new Label()
            {
                Position = new Vector2(50, 50),
                Text = "Thank you for helping me. Here is your reward.",
                Color = Color.White,
                Visible = true
            };

            ControlManager.Add(l);

            _gold = new Label
            {
                Position = new Vector2(50, 50 + size.Y * 2),
                Text = "",
                Color = Color.Yellow,
                TabStop = false,
                Visible = true
            };

            ControlManager.Add(_gold);

            _exp = new Label
            {
                Position = new Vector2(50, 50 + FontManager.GetFont("testfont").LineSpacing *
```

```
3),
                Text = "",
                Color = Color.Yellow,
                TabStop = false,
                Visible = true
            };

            ControlManager.Add(_exp);

            label.Selected += Label_Selected;
        }

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

            _exp.Text = $"You have gained {_reward.Experience} experiene.";
            _gold.Text = $"You received {_reward.Gold} gold.";

            if (InputHandler.KeyReleased(Keys.Escape) ||
                InputHandler.KeyReleased(Keys.Space))
            {
                StateManager.PopState();
            }

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            Rectangle r = new Rectangle(50, 400, 64, 64);

            int _width = 50;

            GameRef.SpriteBatch.Begin();

            base.Draw(gameTime);

            ControlManager.Draw(GameRef.SpriteBatch);

            foreach (var i in _reward.Items)
            {
                GameItem item = GameItemManager.GetItem(i);

                item.Draw(GameRef.SpriteBatch, r);

                r.X += 72;
                _width += 72;

                if (_width >= GameRef.ScreenRectangle.Width - 50)
                {
                    _width = 50;
                    r.Y += 72;
                }
            }

            GameRef.SpriteBatch.End();
        }

        private void Label_Selected(object sender, EventArgs e)
```

```
        {
            StateManager.PopState();
        }
    }
}
```

I added using statements to bring various classes from the MGRpgLibrary and MonoGame framework into scope. Because it is a game screen in inherits from the BaseGameState class. It has three fields: _reward, _gold, and _exp. The first is the reward being displayed. The next two are Labels that draw the gold from the reward and experience from the reward. There is the default constructor that calls the constructor of the base class. The first method, SetReward, requires a Reward parameter and sets the _reward field to the reward passed in.

In the LoadContent method I call the LoadContent method of the base class to create a ControlManager that will be used for drawing text. I create a Vector2 and use the static MeasureString method of the font from the font manager to center the word Close horizontally on the screen. I then create a LinkLabel so that the screen can be exited. I position centered horizontally and just above the bottom of the screen. I set the text to Close and the Visible property to true. I then added it to the control manager. Next I create a Label that displays a thank you message. It is positioned at (50, 50). I set the Text to the message, color to White, Visible to true and TabStop to false. The control is then added to the control manager. If you wanted to display a different message when completing a quest, you could add another field for the message and set it like I do for gold and experience. The control is then added to the control manager. Next a create the _gold Label and set its propertied, then add it to the control manager. Same for the _exp Label. Finally, I wire an event handler for the Close LinkLabel.

In the Update method I call the Update method of the ControlManager so that it will handle the input for the controls. I then set the experience received and gold received, using string interpolation. As well as closing the screen when the link label is selected, I close the screen if the space bar or escape keys are pressed.

In the Draw method there is a Rectangle that is the destination to draw any items that are dropped. I set it just below the middle of the screen and the size of the items to 64 pixels by 64 pixels. There is than a local variable _width that is how far across the screen we are drawing an item, in case there is ever a quest where the player receives a lot of items. I call Begin on the SpriteBatch to start rendering and the Draw method of the base clas. I call the Draw method of the ControlManager property to render the controls.

What is next is the semi-interesting part. There is a foreach loop that loops over all of the items in the reward. I get a GameItem using the GetItem methos of the GameItemManager. I call its Draw method. I then increment _width by 72 pixels and X property of the rectangle by 72. If the _width is greater than or equal the width of the screen - 50, I reset it to 50 and increment the Y property of the rectangle by 72. Finally I call End to end rendering and draw the scene.

The last method is the event handler for the close LinkLabel. It just exits the scene by popping the state off of the stack.

The next step is, of course, to add the state to the game. Add the following property to the Game1

class and update its constructor as follows.

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

public Game1()
{
    _graphics = new GraphicsDeviceManager(this);
    ScreenRectangle = new Rectangle(
        0,
        0,
        ScreenWidth,
        ScreenHeight);
    IsMouseVisible = true;

    Content.RootDirectory = "Content";

    Components.Add(new InputHandler(this));

    _gameStateManager = new GameStateManager(this);
    Components.Add(_gameStateManager);

    _ = new TextureManager();

    TitleScreen = new TitleScreen(this, _gameStateManager);
    StartMenuScreen = new StartMenuScreen(this, _gameStateManager);
    GamePlayScreen = new GamePlayScreen(this, _gameStateManager);
    CharacterGeneratorScreen = new CharacterGeneratorScreen(this, _gameStateManager);
    SkillScreen = new SkillScreen(this, _gameStateManager);
    LoadGameScreen = new LoadGameScreen(this, _gameStateManager);
    ConversationScreen = new ConversationScreen(this, _gameStateManager);
    ShopScreen = new ShopState(this, _gameStateManager);
    InventoryScreen = new InventoryScreen(this, _gameStateManager);
    CombatScreen = new CombatScreen(this, _gameStateManager);
    GameOverScreen = new GameOverScreen(this, _gameStateManager);
    LootScreen = new LootScreen(this, _gameStateManager);
    StatsScreen = new StatsScreen(this, _gameStateManager);
    LevelScreen = new LevelScreen(this, _gameStateManager);
    QuestCompleteState = new QuestCompleteState(this, _gameStateManager);

    _gameStateManager.ChangeState(TitleScreen);

    IsFixedTimeStep = false;
    _graphics.SynchronizeWithVerticalRetrace = false;
}
```

Nothing you haven't seen before. There is a property for the state with a public get accessor and a private set accessor. The constructor just creates a new instance.

There was a bug that I was mentioned. What happens is if you turn in a quest and level up, the level up screen is drawn over the quest complete screen. I worked around that by move the call to HandleConversation after the code that checks for levelling up. That way the reward screen is drawn over the level up screen. When you dismiss the level up screen it goes to the level up screen, and then to the conversation screen. Replace the Update method of the GamePlayScreen class with the following version.

```
public override void Update(GameTime gameTime)
{
    if (targeting)
    {
        List<BaseEffect> effects = currentActivation is Spell
            ? ((Spell)currentActivation).Effects
            : ((Talent)currentActivation).Effects;

        if (effects[0].TargetType == TargetType.Self)
        {
            targeting = false;
            ActivateSelfEffect(effects);
            return;
        }

        if (InputHandler.KeyPressed(Keys.Escape))
        {
            targeting = false;
        }

        if (InputHandler.CheckMousePress(MouseButton.Left))
        {
            castTime = 0;

            if (castTime >= currentActivation.CastTime)
            {
                MobLayer mobLayer = (MobLayer)World.CurrentMap.Layers.Find(x => x is MobLayer);
                Vector2 mouse = InputHandler.MouseAsVector2;
                float scale = (float)currentActivation.AreaOfEffect / target.Width;

                Vector2 targetPosition = Player.Camera.Position + InputHandler.MouseAsVector2;

                foreach (Mob m in mobLayer.Mobs.Values)
                {
                    float distance = Vector2.Distance(targetPosition, m.Sprite.Center);

                    if (distance > currentActivation.AreaOfEffect / 2)
                    {
                        continue;
                    }

                    ActivateEnemyEffect(effects, m);

                    targeting = false;
                }

            }
        }

        return;
    }

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

    player.Camera.LockToSprite(player.Sprite);

    HandleHotKeyInput();

    HandleMobs(gameTime);
```

```
    if (InputHandler.KeyReleased(Keys.I))
    {
        StateManager.PushState(GameRef.InventoryScreen);
    }

    if (InputHandler.KeyReleased(Keys.C))
    {
        StateManager.PushState(GameRef.StatsScreen);
        Visible = true;
    }

    if (Player.Character.Entity.Level < Mechanics.Experiences.Length)
    {
        if (Player.Character.Entity.Experience >=
Mechanics.Experiences[Player.Character.Entity.Level])
        {
            Player.Character.Entity.LevelUp();
            StateManager.PushState(GameRef.LevelScreen);
            Visible = true;
        }
    }

    HandleConversation();

    if (InputHandler.CheckMousePress(MouseButton.Left) && playerTimer > 0.25 && !
playerAttacking)
    {
        playerAttacking = true;
        playerTimer = 0;

        if (player.Sprite.CurrentAnimation == AnimationKey.Up)
        {
            attackDirection = 0;
        }
        else if (player.Sprite.CurrentAnimation == AnimationKey.Right)
        {
            attackDirection = 1;
        }
        else if (player.Sprite.CurrentAnimation == AnimationKey.Down)
        {
            attackDirection = 2;
        }
        else
        {
            attackDirection = 3;
        }
    }

    if (playerTimer >= 0.25)
    {
        playerAttacking = false;
    }

    playerTimer += gameTime.ElapsedGameTime.TotalSeconds;

    if (player.Character.Entity.Health.CurrentValue <= 0)
    {
        StateManager.PushState(GameRef.GameOverScreen);
    }
```

```
    if (player.Character.Entity.Health.CurrentValue <
player.Character.Entity.Health.MaximumValue)
    {
        _healthTimer += gameTime.ElapsedGameTime.TotalSeconds;

        if (_healthTimer > 1)
        {
            _healthTimer = 0;
            player.Character.Entity.Health.Heal(2);
            player.Character.Entity.Mana.Heal(2);
            player.Character.Entity.Stamina.Heal(2);
        }
    }

    base.Update(gameTime);
}
```

Essentially the same code. Only difference is the position of the HandleConversation method. Speaking of that method, it is time to update it for triggering the quest reward screen. Replace that 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].CheckFinished();

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

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

                                foreach (var s in r.Items)
                                {
                                    GameItem item = GameItemManager.GetItem(s);

                                    if (item != null)
                                    {
```

```
                                player.Backpack.AddItem(item);
                            }
                        }

                        npc.Quests.RemoveAt(0);

                        GameRef.QuestCompleteState.SetReward(r);
                        StateManager.PushState(GameRef.QuestCompleteState);
                        return;
                    }
                }

                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);
            }
        }
    }
}
```

The change is now I only test to see if the quest count is greater than zero. After doing that, I call the new CheckFinished method on the quest. It flows as before until I get to the loop where I loop over the item collection for the reward. I call the GetItem method on the GameItemManager class to get the item. If null is not returned the item is added to the player's backpack. After removing the quest from the NPC I call the SetReward method of the QuestCompleteState. Then, I push that state onto the stack of states. I then return out of the method as I don't want to talk with the NPC.

The last thing that I'm going to tackle in the tutorial is the conversation bug. It is a relatively easy fix. What you want to do it when the conversation is changed, set the CurrrentConversation property of the NPC to the new conversation. You can do that by passing in the name of the new conversation as a parameter to the SetConvestation method. You have to do it in two place. The first is when the Change action is selected. Also, you need to do it when a Quest action is selected. Replace 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.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]);
                }
                npc.SetConversation(conversation.CurrentScene.OptionScene);
                conversation =
                conversations.GetConversation(conversation.CurrentScene.OptionScene);
                conversation.StartConversation();
                break;
            case ActionType.Change:
                npc.SetConversation(conversation.CurrentScene.OptionScene);
                conversation =
                conversations.GetConversation(conversation.CurrentScene.OptionScene);
                conversation.StartConversation();
                break;
            case ActionType.End:
                StateManager.PopState();
                break;
        }
    }

    base.Update(gameTime);
}
```

Now, this would work. However, the conversation we created is just a single conversation. It will repeat over and over again. We need to update the conversations to have multiple conversations. Replace the CreateConversations method of the GamePlayScreen with the following version.

If you build and run now everything works as designed. You can speak to NPCs and conversations will move from conversation to conversation. You can accept and fill quests. When you speak the the person giving the quest you get your reward.

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!