Eyes of the Dragon Tutorials
Part 42
Merchants

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.

Be forewarned; this is going to be a long tutorial, but a necessary one. We have our item classes, but currently, they are not available to the player. In this tutorial, I will be adding a merchant to the game for the player to buy and sell items.

So, let's get started. First, I made some changes to the ItemManager class. I made everything static so it can be accessed without an instance of it. Replace the ItemManager class with the following code.

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

namespace RpgLibrary.ItemClasses
{
    public class ItemManager
    {
        #region Fields Region

        static readonly Dictionary<string, Weapon> weapons = new Dictionary<string, Weapon>();
        static readonly Dictionary<string, Armor> armors = new Dictionary<string, Armor>();
        static readonly Dictionary<string, Shield> shields = new Dictionary<string, Shield>();

        #endregion

        #region Keys Property Region

        public static Dictionary<string, Weapon>.KeyCollection WeaponKeys
        {
            get { return weapons.Keys; }
        }

        public static Dictionary<string, Armor>.KeyCollection ArmorKeys
        {
            get { return armors.Keys; }
        }

        public static Dictionary<string, Shield>.KeyCollection ShieldKeys
        {
            get { return shields.Keys; }
        }

        #endregion

        #region Constructor Region
        public ItemManager()
```

```
{
}

#endregion

#region Weapon Methods

public static void AddWeapon(Weapon weapon)
{
    if (!weapons.ContainsKey(weapon.Name))
    {
        weapons.Add(weapon.Name, weapon);
    }
}

public static Weapon GetWeapon(string name)
{
    if (weapons.ContainsKey(name))
    {
        return (Weapon)weapons[name].Clone();
    }

    return null;
}

public static bool ContainsWeapon(string name)
{
    return weapons.ContainsKey(name);
}

#endregion

#region Armor Methods

public static void AddArmor(Armor armor)
{
    if (!armors.ContainsKey(armor.Name))
    {
        armors.Add(armor.Name, armor);
    }
}

public static Armor GetArmor(string name)
{
    if (armors.ContainsKey(name))
    {
        return (Armor)armors[name].Clone();
    }

    return null;
}

public static bool ContainsArmor(string name)
{
    return armors.ContainsKey(name);
}

#endregion

#region Shield Methods
```

```
        public static void AddShield(Shield shield)
        {
            if (!shields.ContainsKey(shield.Name))
            {
                shields.Add(shield.Name, shield);
            }
        }

        public static Shield GetShield(string name)
        {
            if (shields.ContainsKey(name))
            {
                return (Shield)shields[name].Clone();
            }

            return null;
        }

        public static bool ContainsShield(string name)
        {
            return shields.ContainsKey(name);
        }

        #endregion
    }
}
```

As I mentioned, everything is the same other than everything is static. The functionality is the same. There are fields to hold the items, properties to expose their keys, and Add{item}, Get{item}, and Contains{item} methods where {item} is the type of item.

I also update the Backpack class. Replace the code of the Backpack class with the following.

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

namespace MGRpgLibrary.ItemClasses
{
    public class Backpack
    {
        #region Field Region

        readonly List<GameItem> items;

        #endregion

        #region Property Region

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

        public int Capacity
```

```
        {
            get { return items.Count; }
        }

        #endregion

        #region Constructor Region

        public Backpack ()
        {
            items = new List<GameItem>();
        }

        #endregion

        #region Method Region

        public GameItem GetItem(string name)
        {
            GameItem item = items.Find(x => x.Item.Name == name);

            if (item != null)
            {
                items.Remove(item);
            }

            return item;
        }

        public BaseItem PeekItem(string name)
        {
            GameItem item = items.Find(x => x.Item.Name == name);

            return item.Item;
        }

        public void AddItem(GameItem gameItem)
        {
            items.Add(gameItem);
        }

        public void RemoveItem(GameItem gameItem)
        {
            items.Remove(gameItem);
        }

        #endregion

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

I added a field List<GameItem> to hold the items in the Backpack. There currently is no stacking; we will get to that in a future tutorial. Instead, there are just individual items. There is a read-only property to expose the value of the field. In the constructor, I initialize the field.

There is a method to get an item from the Backpack. I use the Find method with a lambda that checks

if there is an item with the name passed in. If there is, I remove the item from the Backpack. I then return either null or the item if it existed in the Backpack.

There is another method PeekItem returns a BaseItem based on the name passed in. Again, I use the Find method of the List<T> class using the same expression to check whether an item with that name exists. I then return the BaseItem from the GameItem.

I added constructors to the Armor, Shield, and Weapon classes. These constructors take {item}Data parameters where {item} is the type of item. They work the same as the other constructors. They initialize the values and call the base constructor. Add these constructors to the appropriate classes.

```
public Armor(ArmorData armorData)
    : base(armorData.Name, armorData.Type, armorData.Price, armorData.Weight,
armorData.AllowableClasses)
{
    Location = armorData.ArmorLocation;
    DefenseValue = armorData.DefenseValue;
    DefenseModifier = armorData.DefenseModifier;
}

public Shield(ShieldData shieldData)
    : base(shieldData.Name, shieldData.Type, shieldData.Price, shieldData.Weight,
shieldData.AllowableClasses)
{
    DefenseValue = shieldData.DefenseValue;
    DefenseModifier = shieldData.DefenseModifier;
}

public Weapon(WeaponData weaponData)
    : base(weaponData.Name, weaponData.Type, weaponData.Price, weaponData.Weight,
weaponData.AllowableClasses)
{
    NumberHands = weaponData.NumberHands;
    AttackValue = weaponData.AttackValue;
    AttackModifier = weaponData.AttackModifier;
    DamageEffects = weaponData.DamageEffectData;
}
```

Some pretty straightforward changes. Instead of requiring individual parameters to set the values, it just requires a single parameter. The call to the base constructor is passed the values from the parameter as well.

I added a using statement, a field, and two properties to the Player class. The first is a Backpack to hold their items, and the second is a field to store their gold carried. Add the following using statement, field, and properties to the Player class.

```
using MGRpgLibrary.ItemClasses;

Backpack backpack = new Backpack();

public Backpack Backpack { get => backpack; set => backpack = value; }

public int Gold { get; set; }
```

I update the EntityType enumeration in the Entity class. I added a new value, Merchant, for Merchant entities. Replace that enumeration with the following.

```
public enum EntityType { Character, NPC, Monster, Creature, Merchant }
```

While we are talking about entities, I added a new class, Merchant. Right-click the CharacterClasses folder in the MGRpgLibrary project, Select Add, and then Class. Name this new class Merchant. The code for that class follows next.

```
using MGRpgLibrary.ItemClasses;
using MGRpgLibrary.SpriteClasses;
using RpgLibrary.CharacterClasses;
using System;
using System.Collections.Generic;
using System.Text;

namespace MGRpgLibrary.CharacterClasses
{
    public class Merchant : Character
    {
        private Backpack _backpack;

        public Backpack Backpack { get => _backpack; set => private _backpack = value; }

        public Merchant(Entity entity, AnimatedSprite sprite) : base(entity, sprite)
        {
            Backpack = new Backpack();
        }
    }
}
```

A pretty straightforward class. It inherits from Character, so it has all the properties, fields, and methods of the Character class. The Backpack field holds the Merchant's invenotry, and there is a property to expose it. The constructor takes the same arguments as a Character: Entity and AnimatedSprite. It calls the base constructor. It then initializes the Backpack.

I added a new region to the Character class. This region has to do with the character's money. Add the following region to the Character class.

```
#region Money Region

private int _gold;

public int Gold { get => _gold; }

public void UpdateGold(int amount)
{
    _gold += amount;
}

#endregion
```

There is a field for the character's gold, _gold. There is a read-only accessor to expose its value. There is also a method, UpdateGold, that updates the character's gold by an amount.

In my Shadow Monsters tutorial series, I added a font manager to manage fonts in the game. Since I took the shop state from that tutorial series, I added one to this tutorial. Right-click the MGRpgLibrary folder, select Add, and then Class. Name this new class FontManager. Here is the code for that class.

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

namespace MGRpgLibrary
{
    public class FontManager
    {
        private static Dictionary<string, SpriteFont> fonts = new Dictionary<string,
SpriteFont>();

        public static void AddFont(string fontName, SpriteFont font)
        {
            if (!fonts.ContainsKey(fontName))
            {
                fonts.Add(fontName, font);
            }
        }

        public static SpriteFont GetFont(string fontName)
        {
            if (fonts.ContainsKey(fontName))
            {
                return fonts[fontName];
            }

            return null;
        }
    }
}
```

The class is meant to be used without an instance, so the field and methods are all static. The field is a Dictionary<string, SpriteFont> to hold the fonts. There is a static method Add that takes as parameters the font's name, which is the key in the Dictionary, and the SpriteFont. If a font with that key does not exist, the font is added to the Dictionary. There is also a method GetFont that takes as a parameter the name of the font to be retrieved. If the key exists, the font with that key is returned. If it does not exist, null is returned.

That brings us to the state for shops. Right-click the GameScreens folder in the EyesOfTheDragon project, select Add, and then Class. Name this new class, ShopState. Here is the code.

```
using MGRpgLibrary;
using MGRpgLibrary.CharacterClasses;
using MGRpgLibrary.ConversationComponents;
using MGRpgLibrary.ItemClasses;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Input;
using RpgLibrary.ItemClasses;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace EyesOfTheDragon.GameScreens
{
    public enum ShopStateType { Buy, Sell, Talk }

    public interface IShopState
    {
        ShopStateType State { get; set; }
        void SetMerchant(Merchant merchant);
    }

    public class ShopState : BaseGameState, IShopState
    {
        private GameScene scene;
        private Merchant merchant;
        private int selected;
        private bool mouseOver;
        private bool isFirst;

        public ShopStateType State { get; set; }

        public ShopState(Game game, GameStateManager manager)
            : base(game, manager)
        {
            State = ShopStateType.Talk;
        }

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

            scene = new GameScene(Game, "basic_scene", "", new List<SceneOption>());
            SceneOption option = new SceneOption("Buy", "Buy", new SceneAction());
            scene.Options.Add(option);

            option = new SceneOption("Sell", "Sell", new SceneAction());
            scene.Options.Add(option);

            option = new SceneOption("Leave", "Leave", new SceneAction());
            scene.Options.Add(option);
        }

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

            scene.Update(gameTime, playerIndexInControl);

            switch (State)
            {
                case ShopStateType.Buy:
                    if (isFirst)
                    {
                        isFirst = false;
                        break;
                    }

                    if (InputHandler.KeyReleased(Keys.Down) ||
```

```
            InputHandler.KeyReleased(Keys.S))
        {
            selected++;
            if (selected >= merchant.Backpack.Items.Count)
            {
                selected = 0;
            }
        }

        if (InputHandler.KeyReleased(Keys.Up) ||
            InputHandler.KeyReleased(Keys.W))
        {
            selected--;
            if (selected < 0)
            {
                selected = merchant.Backpack.Items.Count - 1;
            }
        }


        if (InputHandler.KeyReleased(Keys.Space) ||
            InputHandler.KeyReleased(Keys.Enter) ||
            (InputHandler.CheckMouseReleased(MouseButton.Left) && mouseOver))
        {
            if (selected >= 0 &&
                GamePlayScreen.Player.Gold >=
                merchant.Backpack.PeekItem(
                    merchant.Backpack.Items[selected].Item.Name).Price)
            {
                GamePlayScreen.Player.Backpack.AddItem(
                    merchant.Backpack.GetItem(
                        merchant.Backpack.Items[selected].Item.Name));
                GamePlayScreen.Player.Gold -=
                    merchant.Backpack.PeekItem(
                        merchant.Backpack.Items[selected].Item.Name).Price;
            }
        }
        break;
    case ShopStateType.Sell:
        if (isFirst)
        {
            isFirst = false;
            break;
        }
        if (InputHandler.KeyReleased(Keys.Down) ||
            InputHandler.KeyReleased(Keys.S))
        {
            selected++;

            if (selected >= GamePlayScreen.Player.Backpack.Items.Count)
            {
                selected = 0;
            }
        }

        if (InputHandler.KeyReleased(Keys.Up) ||
            InputHandler.KeyReleased(Keys.W))
        {
            selected--;
            if (selected < 0)
```

```
                    {
                        selected = GamePlayScreen.Player.Backpack.Items.Count - 1;
                    }
                }


                if ((InputHandler.KeyReleased(Keys.Space) ||
                    InputHandler.KeyReleased(Keys.Enter) ||
                    (InputHandler.CheckMouseReleased(MouseButton.Left) && mouseOver)))
                {
                    if (selected >= 0)
                    {
                        GameItem item = GamePlayScreen.Player.Backpack.GetItem(
                            GamePlayScreen.Player.Backpack.Items[selected].Item.Name);
                        GamePlayScreen.Player.Gold += item.Item.Price * 3 / 4;
                    }
                }
                break;
            case ShopStateType.Talk:
                if (InputHandler.KeyReleased(Keys.Space) ||
                    InputHandler.KeyReleased(Keys.Enter))
                {
                    if (scene.SelectedIndex == 0)
                    {
                        isFirst = true;
                        State = ShopStateType.Buy;
                        //selected = -1;
                        return;
                    }

                    if (scene.SelectedIndex == 1)
                    {
                        isFirst = true;
                        State = ShopStateType.Sell;
                        //selected = -1;
                        return;
                    }

                    if (scene.SelectedIndex == 2 && State == ShopStateType.Talk)
                    {
                        StateManager.PopState();
                    }
                }
                break;
        }

        if (InputHandler.CheckMouseReleased(MouseButton.Right) ||
InputHandler.KeyReleased(Keys.Escape))
        {
            switch (State)
            {
                case ShopStateType.Buy:
                case ShopStateType.Sell:
                    State = ShopStateType.Talk;
                    break;
            }
        }
    }

    public override void Draw(GameTime gameTime)
```

```
{
    base.Draw(gameTime);

    GameRef.SpriteBatch.Begin();

    int i = 0;
    Color tint;

    switch (State)
    {
        case ShopStateType.Buy:
            mouseOver = false;
            if (isFirst)
            {
                break;
            }
            GameRef.SpriteBatch.DrawString(
                FontManager.GetFont("testfont"),
                "Item",
                new Vector2(120, 5),
                Color.Red);

            GameRef.SpriteBatch.DrawString(
                FontManager.GetFont("testfont"),
                "Quantity",
                new Vector2(800, 5),
                Color.Red);

            GameRef.SpriteBatch.DrawString(
                FontManager.GetFont("testfont"),
                "Price",
                new Vector2(1100, 5),
                Color.Red);

            foreach (var v in merchant.Backpack.Items)
            {
                tint = Color.White;
                if (i == selected)
                {
                    tint = Color.Red;
                }

                BaseItem item = merchant.Backpack.PeekItem(v.Item.Name);

                if (item != null)
                {
                    Rectangle r = new Rectangle(0, 74 * i + 24, 1200, 64);

                    if (r.Contains(InputHandler.MouseAsPoint))
                    {
                        selected = i;
                        mouseOver = true;
                    }

                    GameRef.SpriteBatch.DrawString(
                        FontManager.GetFont("testfont"),
                        v.Item.Name,
                        new Vector2(120, 74 * i + 45),
                        tint);
```

```
                GameRef.SpriteBatch.DrawString(
                    FontManager.GetFont("testfont"),
                    1.ToString(),
                    new Vector2(800, 74 * i + 45),
                    tint);

                GameRef.SpriteBatch.DrawString(
                    FontManager.GetFont("testfont"),
                    item.Price.ToString(),
                    new Vector2(1100, 74 * i + 45),
                    tint);

                i++;
            }
        }
        break;
    case ShopStateType.Sell:
        mouseOver = false;
        if (isFirst)
        {
            break;
        }
        GameRef.SpriteBatch.DrawString(
            FontManager.GetFont("testfont"),
            "Item",
            new Vector2(120, 5),
            Color.Red);

        GameRef.SpriteBatch.DrawString(
            FontManager.GetFont("testfont"),
            "Quantity",
            new Vector2(800, 5),
            Color.Red);

        GameRef.SpriteBatch.DrawString(
            FontManager.GetFont("testfont"),
            "Price",
            new Vector2(1100, 5),
            Color.Red);

        foreach (var v in GamePlayScreen.Player.Backpack.Items)
        {
            tint = Color.White;

            if (i == selected)
            {
                tint = Color.Red;
            }

            BaseItem item = GamePlayScreen.Player.Backpack.PeekItem(v.Item.Name);

            if (item != null)
            {
                Rectangle r = new Rectangle(0, 74 * i + 24, 1280, 64);

                if (r.Contains(InputHandler.MouseAsPoint))
                {
                    selected = i;
                    mouseOver = true;
                }
```

```
                            GameRef.SpriteBatch.DrawString(
                                FontManager.GetFont("testfont"),
                                v.Item.Name,
                                new Vector2(120, 74 * i + 45),
                                tint);

                            GameRef.SpriteBatch.DrawString(
                                FontManager.GetFont("testfont"),
                                1.ToString(),
                                new Vector2(800, 74 * i + 45),
                                tint);

                            GameRef.SpriteBatch.DrawString(
                                FontManager.GetFont("testfont"),
                                item.Price.ToString(),
                                new Vector2(1100, 74 * i + 45),
                                tint);

                            i++;
                        }
                    }
                    break;
                case ShopStateType.Talk:
                    scene.Draw(gameTime, GameRef.SpriteBatch, null);
                    break;
            }

            GameRef.SpriteBatch.End();
        }

        public void SetMerchant(Merchant merchant)
        {
            this.merchant = merchant;
        }
    }
}
```

Apologies for the wall of code; let's break it down. There is an enumeration that tells us what state the shop is in. The player is either buying, selling, or talking to the Merchant. There is an interface that the class will be implementing. It has a property that gets or sets the state of the shop and a method that sets the Merchant. The class inherits from BaseGameState and implements the interface.

There are five fields in the class. The first, scene, is what is displayed when the player is talking to the Merchant. The merchant field is the Merchant we are talking to. The selected field is the currently selected item for buying or selling. The isFirst field is similar to the frameCount field and is used to prevent bleeding of key/mouse presses. There is also the property from the interface. The constructor sets the state to talk. The LoadContent method creates the scene. I do it a little differently. I make the options and add them to the scene directly rather than create a list of options and the scene.

The Update method calls the Update method of the scene to update it. Next, there is a switch on the State property. If the state is buying, I check the isFirst field is true. If it is, I set it to false and break out of the case. Next, I check if the down or S keys have been pressed. If they have, I move to the next item in the Merchant's backpack, wrapping to the first if we are at the end of the list. I move in the opposite direction if the up or W keys have been pressed. If the space bar or enter keys have been pressed or the mouse is over, and the left button has been pressed, I check to see if there is a selected item and that the player has enough money to buy

the item. If they do, I add the item to the player's Backpack and deduct the price from their gold. The Sell case works similarly but on the player's Backpack instead of the Merchant's Backpack. When selling, I give the player 75% of the item's value. The talk case checks to see if the space bar or enter keys have been pressed. If they have, I call FlushInput to reset the input. If the selected state is 0 or Buy I set isFirst to true and State to Buy, then exit the method. I do something similar for the Sell option. Finally, if we are in the Talk state, I pop the state off of the stack. If the right mouse button or escape key has been pressed, I go to the Talk state.

The Draw method renders the backpacks and the scene. There are two local variables. The first is a counter, and the second determines what color an item in the Backpack is drawn in. There is a switch on the state. In the Buy case mouseOver is set to false. If this is the first time the state is being drawn, break out of the case. Next, I position the title for the screen. I then loop over all of the items in the Merchant's Backpack. The tint is set to white, then if the counter is equal to the selected item, it is set to red. I then peek if the item is in the Backpack. If the item is not null, I create a rectangle around the item. If it contains the mouse selected is set to the counter and mouse over to true. Finally, I draw the item and increment the counter. The Sell option works the same way but on the player's Backpack. The Talk state just draws the scene.

The SetMerchant method just sets the merchant field. Now we need to implement the new state. The first step is to create a property in the Game1 class and then create an instance. Add the following field and property and update the constructor of the Game1 class.

```
public ShopState ShopScreen { get; }
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);

    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);

    _gameStateManager.ChangeState(TitleScreen);

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

The other change that I made was to set IsMouseVisible equal to true. It keeps disappearing, so I added it back in again. While we have the Game1 class open, let's add the font to the FontManager. Change the LoadContent

method to the following.

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

    DataManager.ReadEntityData(Content);
    DataManager.ReadArmorData(Content);
    DataManager.ReadShieldData(Content);
    DataManager.ReadWeaponData(Content);
    DataManager.ReadChestData(Content);
    DataManager.ReadKeyData(Content);
    DataManager.ReadSkillData(Content);

    FontManager.AddFont("testfont", Content.Load<SpriteFont>("Fonts/scenefont"));
}
```

Now I'm going to update the GamePlayScreen. What needs to happen is to check for merchants when space is pressed and if it is, then trigger the merchant scene. Update the Update method of the GamePlayScreen to the following.

```
public override void Update(GameTime gameTime)
{
    world.Update(gameTime);
    player.Update(gameTime);
    player.Camera.LockToSprite(player.Sprite);

    if (InputHandler.KeyReleased(Keys.Space) ||
        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);
                    }
                }
            }
        }
    }

    base.Update(gameTime);
}
```

What happens is the code check to see if the spacebar was pressed. If it was is checks the characters on the map to see if they are close enough to the player to start a conversation. If the character is a NonPlayerCharcter a conversation is started. If the character is a Merchant, the shop scene is pushed onto the stack, and the SetMerchant method is called on the state passing in the Merchant.

The final changes for this tutorial are in the CharacterGeneratorScene class. I did two things. The first is that in the CreatePlayer method, I give the player some gold. The second is in the LoadWorld method, I create a merchant through code and add it to the map. First, update the CreatePlayer method to the following.

```
private void CreatePlayer()
{
    int gender = genderSelector.SelectedIndex < 2 ? genderSelector.SelectedIndex : 1;

    AnimatedSprite sprite = new AnimatedSprite(
        characterImages[gender, classSelector.SelectedIndex],
        AnimationManager.Instance.Animations);
    EntityGender g = EntityGender.Unknown;

    switch (genderSelector.SelectedIndex)
    {
        case 0:
            g = EntityGender.Male;
            break;
        case 1:
            g = EntityGender.Female;
            break;
        case 2:
            g = EntityGender.NonBinary;
            break;
    }

    Entity entity = new Entity(
        nameSelector.SelectedItem,
        DataManager.EntityData[classSelector.SelectedItem],
        g,
        EntityType.Character);

    foreach (string s in DataManager.SkillData.Keys)
    {
        Skill skill = Skill.FromSkillData(DataManager.SkillData[s]);
        entity.Skills.Add(s, skill);
    }

    Character character = new Character(entity, sprite);
    GamePlayScreen.Player = new Player(GameRef, character);
    GamePlayScreen.Player.Gold = 200;
}
```

Now, the LoadWorld method. Replace the old LoadWorld method with this new code.

```
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);


    string[] fileNames = Directory.GetFiles(
        Path.Combine("Content/Game/Items", "Armor"),
        "*.xnb");

    foreach (string a in fileNames)
    {
        string path = "Game/Items/Armor/" + Path.GetFileNameWithoutExtension(a);

        ArmorData armorData = Game.Content.Load<ArmorData>(path);
        ItemManager.AddArmor(new Armor(armorData));
    }

    fileNames = Directory.GetFiles(
        Path.Combine("Content/Game/Items", "Shield"),
        "*.xnb");

    foreach (string a in fileNames)
    {
        string path = "Game/Items/Shield/" + Path.GetFileNameWithoutExtension(a);

        ShieldData shieldData = Game.Content.Load<ShieldData>(path);
        ItemManager.AddShield(new Shield(shieldData));
    }

    fileNames = Directory.GetFiles(
        Path.Combine("Content/Game/Items", "Weapon"),
        "*.xnb");

    foreach (string a in fileNames)
    {
        string path = "Game/Items/Weapon/" + Path.GetFileNameWithoutExtension(a);

        WeaponData weaponData = Game.Content.Load<WeaponData>(path);
        ItemManager.AddWeapon(new Weapon(weaponData));
    }

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

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

    foreach (var c in charData.Characters)
    {
        Character character;

        if (c.Value is NonPlayerCharacterData)
        {
```

```
            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(
                    ((NonPlayerCharacterData)c.Value).CurrentConversation);
            }

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

    map.AddLayer(characterLayer);

    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(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("eliza1");
    //world.Levels[world.CurrentLevel].Characters.Add(npc);

    s = new AnimatedSprite(
        GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
        AnimationManager.Instance.Animations);
    s.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(new GameItem(ItemManager.GetWeapon("Long Sword"), items, new
Rectangle(0, 0, 16, 16)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetWeapon("Short Sword"), items, new
Rectangle(16, 0, 16, 16)));

    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;

    CreateConversation();

    //
((NonPlayerCharacter)world.Levels[world.CurrentLevel].Characters[0]).SetConversation("eliza1");
}
```

The first thing I do is load in the armor, shields, and weapons. I do that by getting all of the files in either the armor, shield, or weapon folders with the XNB extension. I then loop over all of those files. I build the path to the file without extension. I then use the ContentManager to load the file and add it to the ItemManager.

After creating a test NPC, which I no longer add to the map, I create a duplicate AnimatedSprite because I didn't have another sprite handy that I could use. I then create duplicate EntityData for the Merchant. I create an entity, Barbra, using the entity data passing in Merchant for the entity type. I then create a Merchant. I also load a texture for the items that I will add in a minute. I add a Long Sword and Short Sword to the Merchant's Backpack next. I just used two random source rectangles for the items in this tutorial. I then use the Find method of the List<T> class to find the character layer. I add the Merchant to the layer at tile (10, 0).

The last thing I'm going to do in this tutorial is to add the item sheet texture. The texture I used a CC-BY-3.0-SA. You can find it on the creator's page on OpenGameArt.org. Once you have downloaded it, use the MonoGame Pipeline Tool to add it to the ObjectSprites folder.

Though not as long as I had thought initially, I'm going to wrap up the tutorial here because I don't want to do something new at this point. 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