Eyes of the Dragon Tutorials
Part 44
Equipment – 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.

This tutorial will continue from the last tutorial which covered rendering the player's backpack. It will cover actually equipping items from inventory on the player character. It will be on the entity that is part of the player character. The reason being that both characters and mobs will require access to items, and they all contain entities. Let's get started.

The first thing that I want to do is change the IsEquipped property of the BaseItem class. Right now it has a protected set accessor. I want to change it to be a public set accessor. The reason I want to do this is I'm going to move equipping items entirely into the Entity class. I still want the item to have an IsEquipped property, though. That is because it will be required for rendering and in the combat engine. Change the IsEquipped property of the BaseItem class to the following code.

```
        public bool IsEquiped
        {
            get { return equipped; }
            set { equipped = value; }
        }
```

The next thing that I am going to do is add an Equip method to the Entity class. This method will be called when the player clicks on an item in the backpack. Add the following method to the Entity class.

```
public void Equip(GameItem item)
{
    if (!item.Item.AllowableClasses.Contains(EntityClass))
    {
        return;
    }

    if (item.Item is Weapon weapon)
    {
        if (mainHand == null)
        {
            mainHand = item;
            item.Item.IsEquiped = true;
        }
        else
        {
            if (mainHand != item)
            {
                mainHand.Item.IsEquiped = false;
                mainHand = item;
                mainHand.Item.IsEquiped = true;
            }
        else
        {
            mainHand = null;
            item.Item.IsEquiped = false;
        }
    }

    if (weapon.NumberHands == ItemClasses.Hands.Both && offHand != null)
    {
        offHand.Item.IsEquiped = false;
        offHand = null;
    }
}

if (item.Item is Shield shield)
{
    if (offHand != null)
    {
        offHand.Item.IsEquiped = false;
    }

    offHand = item;
    offHand.Item.IsEquiped = true;
}

if (item.Item is Armor armor)
{
    if (armor.Location == ArmorLocation.Body)
    {
        if (body != null)
        {
            body.Item.IsEquiped = false;
        }

        body = item;
        body.Item.IsEquiped = true;
    }

    if (armor.Location == ArmorLocation.Head)
    {
        if (head != null)
        {
            head.Item.IsEquiped = false;
        }

        head = item;
        head.Item.IsEquiped = true;
    }

    if (armor.Location == ArmorLocation.Hands)
    {
        if (hands != null)
        {
            hands.Item.IsEquiped = false;
        }

        hands = item;
        hands.Item.IsEquiped = true;
    }
```

```
        if (armor.Location == ArmorLocation.Head)
        {
            if (feet != null)
            {
                feet.Item.IsEquiped = false;
            }

            feet = item;
            feet.Item.IsEquiped = true;
        }
    }
}
```

As you can see, the method requires a GameItem parameter. This is the item that the player is going to try and equip on their character. I first check to see if the class of the entity is not in the list of allowed classes. If it is not, I exit out of the method straight away.

Next I use pattern matching to test if the item is a weapon and assign that to a local variable. If there is nothing in the entity's main hand I set the main hand to be the weapon and I set it's IsEquipped property to be true. If it is not null, I check to see if the weapon in the main hand is not the weapon being equipped. If not, I set the IsEuipped property to false, set the item in the main hand to be the item that was passed in. Finally I set IsEquipped property of the main hand weapon to true. If it was the weapon that is equipped, I set the equipped weapon to null and the IsEquipped property of the weapon to false. The next check checks to see if the weapon is a two handed weapon and that there is something in the off hand. If those conditions are true, I set the IsEquipped property of the off hand to false and then set it to null. Currently I am not handling off hand weapons and dual wielding. I will add this in a future tutorial.

The next test is to check if the item is a shield and create a local shield variable. I technically didn't have to do this, but I did it in case I need it in the future. Like in the weapon code I check to see if the off hand is not null. If it is not null, I set the IsEquipped property to false. I then set the off hand to be the item passed in and set the IsEquipped property to true.

The last check is to see if the item is armor. I then check to see if the location for the armor is the body. If it is the body, I check to see if the body is not null. If it is not null, I set the IsEquipped property to false. I then set body to be the item and the IsEquipped property to true. The cases for the head, hands and feet work the exact same way. So, I am not going to go into details on those cases.

That is all there is to equipping items on an entity. Now we need to update the InventoryScreen to render the backpack and the equipped items. Since there were a number of modifications I will give you the code for the entire class. Replace the InventoryScreen class with the following code.

```
using EyesOfTheDragon.Components;
using MGRpgLibrary;
using MGRpgLibrary.ItemClasses;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using System.Text;
```

```
namespace EyesOfTheDragon.GameScreens
{
    public class InventoryScreen : BaseGameState
    {
        private Texture2D _background;
        private Texture2D _disabledTexture;
        private Rectangle _destination;
        private List<Rectangle> _backpackDestinations = new List<Rectangle>();
        private Rectangle _headGear;
        private Rectangle _bodyGear;
        private Rectangle _handGear;
        private Rectangle _footGear;
        private Rectangle _neckGear;
        private Rectangle _mainGear;
        private Rectangle _offGear;
        private Rectangle _fingerGear;

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

        public override void Initialize()
        {
            int x = 622, y = 96;

            for (int i = 0; i < 30; i++)
            {
                _backpackDestinations.Add(new Rectangle(x, y, 106, 78));

                x += 123;

                if (x >= 1220)
                {
                    x = 622;
                    y += 78;
                }
            }

            _headGear = new Rectangle(55, 157, 101, 73);
            _bodyGear = new Rectangle(55, 247, 101, 73);
            _handGear = new Rectangle(55, 337, 101, 73);
            _footGear = new Rectangle(55, 427, 101, 73);
            _neckGear = new Rectangle(500, 157, 101, 73);
            _mainGear = new Rectangle(500, 247, 101, 73);
            _offGear = new Rectangle(500, 337, 101, 73);
            _fingerGear = new Rectangle(500, 427, 101, 73);

            base.Initialize();
        }

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

            _background = Game.Content.Load<Texture2D>(@"GUI/inventory-gui");
            _destination = new Rectangle(0, 0, Game1.ScreenWidth, Game1.ScreenHeight);
            _disabledTexture = Game.Content.Load<Texture2D>(@"GUI/disabled");
        }
        public override void Update(GameTime gameTime)
        {
            if (InputHandler.CheckMouseReleased(MouseButton.Right) ||
InputHandler.KeyReleased(Keys.Escape))
            {
                StateManager.PopState();
            }

            if (InputHandler.CheckMouseReleased(MouseButton.Left))
            {
                if (new Rectangle(1175, 45, 42, 32).Contains(InputHandler.MouseAsPoint))
                {
                    StateManager.PopState();
                }

                for (int i = 0; i < _backpackDestinations.Count && i <
GamePlayScreen.Player.Backpack.Items.Count; i++)
                {
                    if (_backpackDestinations[i].Contains(InputHandler.MouseAsPoint))
                    {

GamePlayScreen.Player.Character.Entity.Equip(GamePlayScreen.Player.Backpack.Items[i]);
                        break;
                    }
                }
            }
            base.Update(gameTime);
        }

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

            GameRef.SpriteBatch.Begin();

            GameRef.SpriteBatch.Draw(_background, _destination, Color.White);

            int c = 0;

            foreach (GameItem item in GamePlayScreen.Player.Backpack.Items)
            {
                item.Draw(GameRef.SpriteBatch, _backpackDestinations[c]);

                if (item.Item.IsEquiped)
                {
                    GameRef.SpriteBatch.Draw(_disabledTexture, _backpackDestinations[c],
Color.White);
                }

                c++;
                if (c >= _backpackDestinations.Count)
                {
                    break;
                }
            }

            Player p = GamePlayScreen.Player;

            if (p.Character.Entity.MainHand != null &&
p.Character.Entity.MainHand.Item.IsEquiped)
```

```
            {
                p.Character.Entity.MainHand.Draw(GameRef.SpriteBatch, _mainGear);
            }

            if (e.OffHand != null)
            {
                e.OffHand.Draw(GameRef.SpriteBatch, _offGear);
            }

            if (e.Head != null)
            {
                e.Head.Draw(GameRef.SpriteBatch, _headGear);
            }

            if (e.Body != null)
            {
                e.Body.Draw(GameRef.SpriteBatch, _bodyGear);
            }

            if (e.Hands != null)
            {
                e.Hands.Draw(GameRef.SpriteBatch, _handGear);
            }

            if (e.Feet != null)
            {
                e.Feet.Draw(GameRef.SpriteBatch, _footGear);
            }

            GameRef.SpriteBatch.End();
        }
    }
}
```

The first change was the addition of a field _disabledTexture that is a semi-transparent black texture that will be drawn over equipped items. You can download my texture from my blog at the following link: https://cynthiamcmahon.ca/blog/downloads/disabled.png. Once you have downloaded it open the MonoGame Content Pipeline tool, right click the GUI folder, and select Add Existing Item. Navigate to the disabled.png file and add it to the folder. Save the project and close the tool. I added eight Rectangle fields. One for the head armor, body armor, hand armor, foot armor, necklace, main hand, off hand and ring. I haven't added necklaces and rings yet but I will be so I included the field. That, and there are eight boxes to be filled. In the Initialize method I create the destination rectangles for the new fields. I calculated them using a screen shot of the inventory screen and an image editor. In the LoadContent method I load the disabled texture.

In the Update method I check to see if the left mouse button has been released. If it has I check to see if a rectangle around the X in the background image contains the mouse as a point. If it does, I pop the state off the stack. Next is a for loop that loops over the destination rectangles in the backpack up to the number of items in the backpack. If the destination rectangle contains the mouse location as a point I call the Equip method of the entity passing in the item. I then break out of the loop.

The Draw method works similarly as before. The first change is after drawing the item it checks to see if the item is equipped. If it is equipped I draw the disabled texture at the destination rectangle. The next thing I do is assign the entity for the player to a local variable so that I don't have to type it eight times, well six because currently there are no rings or necklaces. After assigning it there are a series of if statements that check to see if an item is equipped at a certain location. If an item is equipped at that location I draw the item at the appropriate destination.

I thought that this part of the tutorial series was going to take longer than it did. Initially I was going to drag items off the backpack on to the squares. Instead I decided to go with the click approach. Also, I didn't handle dual weapons. Still, I'm going to wrap up the tutorial here because I don't want to start 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