Eyes of the Dragon Tutorials 4.0

Part 14

Mish Mash

I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. 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 4.0 page of my web blog. I will be making the project available on GitHub here. It will be included on the page that links to the tutorials.

This is part fourteen of a series of tutorials I plan to write on creating a role-playing game with MonoGame. I've worked on similar tutorials using XNA and the past version of MonoGame. In the process of writing more tutorials, I discovered better ways of doing some things and had to go back to fix things. I'm hoping in this series to avoid making those same mistakes. Also, I am going to make the game cross-platform. In my previous tutorials, they were focused on Windows only. I want to open things up for macOS, Linux, Android, iOS, and any platform MonoGame supports.

For this tutorial, we are going to need a sprite for a mob. Fortunately, OpenGameArt.org has plenty of resources that we can use. I chose a bat by bagzie. You can download the sprite with the following link: https://opengameart.org/content/bat-sprite. After you have downloaded it, you need to add it as content to the SharedProject. Double-click the Content folder in the SharedProject to reveal the Content.mgcb node. Double-click that to open the MGCB Editor. Right-click the Content node, select Add and then New Folder. Name this new folder Mobs. Now, right-click the Mobs folder, select Add and then Existing item. Navigate to where you downloaded the bat and close the editor.

Before I get to mobs, I want to add a random number generator to the Helper class. It will just be a static field that is exposed by a property. Replace the Helper class with the following version.

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

namespace SharedProject
{
    public static class Helper
    {
        private static readonly Random _random = new();

        public static Vector2 NearestInt(Vector2 vector2)
        {
            return new((int)vector2.X, (int)vector2.Y);
        }

        public static Point V2P(Vector2 vector2)
        {
            return new((int)vector2.X, (int)vector2.Y);
        }
```

```
        public static Random Random
        {
            get { return _random; }
        }
    }
}
```

Mobs will be created from comma-separated value strings. This will allow for easy editing of mobs. The only challenging part will be creating animated sprites on the fly. It is not impossible, though. We just need the texture and the animations. Let's start with the base class for mobs. After this, I will add the method that we will be using to create a sprite on the fly. Right-click the SharedProjects project in the solution explorer, select Add and the New Folder. Name this new folder Mobs. Right-click the Mobs folder, select Add and then Class. Name this new class Mob. Here is the base code for the Mob class.

```
using Assimp.Configs;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using RpgLibrary.Characters;
using SharedProject.Sprites;
using System;
using System.Collections.Generic;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Text;

namespace SharedProject.Mobs
{
    public class Mob : ICharacter
    {
        private Mob() { }
        private AnimatedSprite AnimatedSprite { get; set; }

        private string _name;
        private string _description;

        public string Name { get => _name; private set => _name = value; }
        public string Description { get => _description; private set => _description =
value; }
        public int Strength { get; set; }
        public int Perception { get; set; }
        public int Endurance { get; set; }
        public int Charisma { get; set; }
        public int Intellect { get; set; }
        public int Agility { get; set; }
        public int Luck { get; set; }

        public AttributePair Health { get; set; }
        public AttributePair Mana { get; set; }

        public int Gold { get; set; }
        public int Experience { get; set; }
        public bool Enabled { get; set; }
        public bool Visible { get; set; }
        public Vector2 Position { get; set; }
```

```
public Point Tile { get; set; }

public bool RollD10(string attribute)
{
    PropertyInfo info = this.GetType().GetProperty(attribute);

    if (info != null )
    {
        if (info.GetType() == typeof(int))
        {
            int roll = Helper.Random.Next(1, 11);

            if (roll == 10)
            {
                return false;
            }

            if (roll == 1)
            {
                return true;
            }

            if (int.TryParse(info.GetValue(this).ToString(), out int value))
            {
                return (roll <= value);
            }
        }
    }

    return false;
}

public virtual void Draw(SpriteBatch spriteBatch)
{
    AnimatedSprite.Draw(spriteBatch);
}

public virtual void Update(GameTime gameTime)
{
    AnimatedSprite.Update(gameTime);
}

public static Mob FromString(string data, ContentManager content)
{
    Mob mob = new();
    string[] parts = data.Split(',');

    foreach (string part in parts)
    {
        string[] pair = part.Split('=');

        PropertyInfo info = mob.GetType().GetProperty(pair[0]);

        if (info != null)
        {
            Type type = info.PropertyType;
            if (type != typeof(AttributePair) &&
                type != typeof(AnimatedSprite) &&
                type != typeof(Vector2) &&
                type != typeof(Point))
```

```
                    {
                        if (typeof(String) == type)
                        {
                            info.SetValue(mob, pair[1], null);
                        }
                        else
                        {
                            info.SetValue(mob, int.Parse(pair[1]), null);
                        }
                    }
                    else if (type == typeof(AttributePair))
                    {
                        info.SetValue(mob, new AttributePair(int.Parse(pair[1])), null);
                    }
                    else if (type == typeof(AnimatedSprite))
                    {
                        CreateAnimatedSprite(content, mob, pair[1]);
                    }
                    else if (type == typeof(Vector2))
                    {
                        string[] vector = pair[1].Split(':');

                        if (float.TryParse(vector[0], out float x) && float.TryParse(vec-
tor[1], out float y))
                        {
                            info.SetValue(mob, new Vector2(x, y));
                        }
                    }
                    else if (type == typeof(Point))
                    {
                        string[] point = pair[1].Split(':');

                        if (int.TryParse(point[0], out int x) && int.TryParse(point[1],
out int y))
                        {
                            info.SetValue(mob, new Point(x, y));
                        }
                    }

                }
            }

            mob.AnimatedSprite.Position = new(mob.Tile.X * Engine.TileWidth, mob.Tile.Y *
Engine.TileHeight);
            return mob;
        }

        private static void CreateAnimatedSprite(ContentManager content, string v)
        {
            throw new NotImplementedException();
        }
    }
}
```

This class implements the ICharacter interface. We do this because we want access to the SPECIAL attributes. It has a private constructor that takes no parameters. This is so that we force the user to use the FromString method. I guess, I could have just added a string parameter to the constructor.

There is a private property for the animated sprite. There are private fields for the name and description of the mob. Next are the properties of the interface.

I added a method, RollD10, that rolls a die against the attribute passed in. Instead of an if statement or switch, I used reflection. If felt better because I was using the same code seven times. The first step is to get the property information. Remember that the attribute is case-sensitive. Next, check to see if the property exists. Now, check to see if it is an integer property. Technically, you could roll against gold and experience this way, but I don't know why you would do that. Now, use the Random property of the Helper class to generate a random number between one and ten. Remember, a roll of ten always fails, and a roll of one always succeeds. Next, parse the value of the property as an integer, then return if the roll is less than or equal to the value. If all of those checks fail, return false.

The Draw method calls the Draw method of the AnimatedSprite and the Update method calls the Update method of the AnimatedSprite.

That brings us to the FromString method. As I mentioned earlier, this could have easily been a constructor. It requires a string to parse and a ContentManager for loading content. The first step is to create a new mob. Now, split the string on the comma character. Now loop over each part. Then, split the part on the equals sign. I used reflection again. I just love it! If there is a property for the part, I get the type of the property, and I check to see that the part is not an AttributePair, AnimatedSprite, Vector2, or Point. Then, if it is a string, I just call SetValue passing in the string. I probably should have check if the second part is an integer rather than just casting it to an integer, but I just parse the second part of the pair as an integer. If it is an AttributePair, I call the SetValue passing in an attribute pair with the second part of the pair parsed as an integer. If it is an AnimatedSprite, Call the CreateAnimatedSprite method passing in the content manager, the mob we are working on, and the second part of the pair.

Vector2s and Points are virtually the same. The steps are to split the second part of the pair on the : character. Then, check to see if parsing both halves of the pairs succeeds using TryParse, capturing the result in x and y variables. Then, call SetValue, passing in a new Vector2 or Point.

The last thing left to do before returning the mob is to set its Position property. I use the Tile property and multiply the X coordinate by the tile width and the Y coordinate by the tile height.

CreateAnimatedSprite is just a method stub at this point. We will flesh it out now. Why did I do it this way, you may be asking? I was trying to spare you long copies/pastes. In actuality, it wasn't as bad as I thought it was going to be. What we need is the name of the texture and then the different animations. That was the tricky part. So, we can't separate items by commas or equal signs. So, we will do it this way. We will split it up using semicolons. The first part will be the name of the texture. The following parts will be separated by colons. The first will be the name of the animation. Next, will be the x-offset, followed by the y-offset, the frame width, the frame height and finally, the frame count. The frame count was included rather than calculated using the width and height of the frames and texture because there is a case where you will have more than one sprite in the same sheet. It is actually more efficient for the graphics card to process large textures rather than loading many small ones. It is even better if they are in powers of two. There is one last addition that I made. I appended

the current animation after the animations. Replace the CreateAnimatedSprite method with this new version.

```
private static void CreateAnimatedSprite(ContentManager content, Mob mob, string v)
{
    string[] parts = v.Split(';');
    Texture2D texture = content.Load<Texture2D>($"Mobs/{parts[0]}");
    Dictionary<string, Animation> animations = new();

    for (int i = 1;  i < parts.Length - 1; i++)
    {
        string[] animationDesc = parts[i].Split(":");

        if (int.TryParse(animationDesc[1], out int xoffset) &&
            int.TryParse(animationDesc[2], out int yoffset) &&
            int.TryParse(animationDesc[3], out int width) &&
            int.TryParse(animationDesc[4], out int height) &&
            int.TryParse(animationDesc[5], out int count))
        {
            Animation animation = new(count, width, height, xoffset, yoffset);

            animations.Add(animationDesc[0], animation);
        }
    }

    mob.AnimatedSprite = new(texture, animations)
    {
        CurrentAnimation = parts[^1],
        IsActive = true,
        IsAnimating = true
    };
}
```

The first step is to split the string on the semicolon. Next, load the texture, using string interpolation rather than string concatenation because it is more efficient. Now, create a dictionary to hold the animations. I now loop over the different animations. The animations are then split into their components using the colon. I then try parsing the individual components. They are in the order of x-offset, y-offset, width, height, and frame count. If all of the values are parsed correctly, I create an animation and add it to the dictionary. After parsing the animations, I create an AnimatedSprite and assign its CurrentAnimation property to the value passed in and IsActive and IsAnimating properties to true.

That went a lot faster than I had anticipated, coming from my old shadow monsters tutorials. I'm not ready to code the combat state yet, but I don't want to end this tutorial here. So, what I'm going to do is add mobs to the map. To do that, I will add a new class that represents an encounter. Encounters are the maps on which combat will be fought. An encounter may have one or more mobs on it. It will also have rewards. To start, we want to add a class to the RpgLibrary project for encounters. Right-click the RpgLibrary project, select Add and then Class. Name this new class Encounter. Here is the code for that class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RpgLibrary.Characters;
```

```
using RpgLibrary.TileEngine;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RpgLibrary
{
    public class Encounter
    {
        public List<ICharacter> Allies { get; set; } = new();
        public List<ICharacter> Enemies { get; set; } = new();
        public TileMap Map { get; set; }

        public bool Alive => Enemies.Count > 0 && Allies.Count > 0;

        public Encounter(ICharacter player)
        {
            Allies.Add(player);
        }

        public void Update(GameTime gameTime)
        {
            Map?.Update(gameTime);

            foreach (var character in Enemies)
            {
                character.Update(gameTime);
            }

            Enemies.RemoveAll(x => x.Health.Current <= 0);
            Allies.RemoveAll(x => x.Health.Current <= 0);
        }

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera)
        {
            Map?.Draw(gameTime, spriteBatch, camera);

            foreach (var character in Enemies)
            {
                character.Draw(spriteBatch);
            }

            foreach (var character in Allies)
            {
                character.Draw(spriteBatch);
            }
        }
    }
}
```

A deceptively simple, yet powerful class. There are two lists of ICharacter. One for the player's side and one for the enemy's side. There is also a TileMap for the encounter to take place on. There is a property that returns if the encounter is "alive", meaning that there is an ally alive and an enemy alive.

There is a single constructor that takes an ICharacter parameter which is the player. The player is then added to the list of allies. The Update method requires a GameTime parameter. If the map is not null,

its Update method is called. It then loops over the enemies and calls their Update method. Similarly, it loops over the allies and calls their Update method. Next, is a little LINQ to remove the dead allies and enemies.

The Draw method takes three parameters, those necessary to render a map. Those are a GameTime, SpriteBatch and Camera. If the Map property is not null, it calls the Draw method of the map. It then loops over the enemies and allies, calling their Draw method.

Before I get to adding encounters to a map, I need to make an update to the ISprite interface that is in the RpgLibrary's Sprites folder. I need to add methods to update and draw a sprite. This is because encounters will be on the map, but we need some way to represent them. So, I will use sprites as a key for a dictionary and update and render the sprite. Replace the ISprite interface with the following.

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

namespace RpgLibrary.Sprites
{
    public interface ISprite
    {
        Vector2 Position { get; set; }
        Vector2 Size { get; set; }
        int Width { get; }
        int Height { get; }
        void Update(GameTime gameTime);
        void Draw(SpriteBatch spriteBatch);
    }
}
```

Now, we need to add encounters to a map. I did that by adding a new layer type, an EncounterLayer. In the RpgLibrary project, right-click the TileEngine folder, select Add and then Class. Name this new class EncounterLayer. Here is the code for that class.

```
using Assimp;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RpgLibrary.Sprites;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RpgLibrary.TileEngine
{
    public class EncounterLayer : ILayer
    {
        public Dictionary<ISprite, Encounter> Encounters { get; private set; } = new();
```

```
        public EncounterLayer() { }

        public void Draw(SpriteBatch spriteBatch, Camera camera, List<Tileset> tilesets)
        {
            foreach (ISprite sprite in Encounters.Keys)
            {
                if (Encounters[sprite].Alive)
                {
                    sprite.Draw(spriteBatch);
                }
            }
        }

        public void Update(GameTime gameTime)
        {
            foreach (ISprite sprite in Encounters.Keys)
            {
                if (Encounters[sprite].Alive)
                {
                    sprite.Update(gameTime);
                }
            }
        }
    }
}
```

Another deceptively unremarkable class. It implements the ILayer class so it can be added to a map easily. That way, its Update and Draw methods will be called behind the scenes. It has a single property that is a Dictionary<ISprite, Encounter>. This, as you may have guessed, holds the encounters on the map. I used an ISprite for the key because it will be drawn on the map to represent one of the mobs that make up the encounter. You can be very deceptive here. You can draw a giant bat representing the encounter but have a red dragon as one of the mobs.

There is a single constructor that takes no parameters. The Draw method takes the three required parameters: SpriteBatch, Camera, and List<Tileset>. It cycles over the key collection of the encounters property. If the encounter is alive, I draw it. I do something similar in the Update method, but instead of calling Draw, I call Update.

Let's add an encounter layer to the map and an encounter. The encounter will take place at tile (5, 5). First, update the using statements of the GamePlayState to the following. They should be as follows.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Rpglibrary.TileEngine;
using RpgLibrary;
using RpgLibrary.Characters;
using RpgLibrary.TileEngine;
using SharedProject.Characters;
using SharedProject.Controls;
using SharedProject.GamesScreens;
using SharedProject.Mobs;
using SharedProject.Sprites;
using SharedProject.StateManagement;
using System;
using System.Collections.Generic;
```

```
using System.Linq;
```

Now, update the LoadContent method to create an EncounterLayer. The code for that method follows next.

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

    _conversationState = Game.Services.GetService<IConversationState>();
    _conversationManager = Game.Services.GetService<IConversationManager>();

    renderTarget = new(GraphicsDevice, Settings.BaseWidth, Settings.BaseHeight);

    Texture2D texture = Game.Content.Load<Texture2D>(@"Tiles/tileset1");

    List<Tileset> tilesets = new()
    {
        new(texture, 8, 8, 32, 32),
    };

    TileLayer layer = new(100, 100);

    map = new("test", tilesets[0], layer);

    Dictionary<string, Animation> animations = new();

    Animation animation = new(3, 32, 32, 0, 0) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkdown", animation);

    animation = new(3, 32, 32, 0, 32) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkleft", animation);

    animation = new(3, 32, 32, 0, 64) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkright", animation);

    animation = new(3, 32, 32, 0, 96) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkup", animation);

    texture = Game.Content.Load<Texture2D>(@"PlayerSprites/femalefighter");

    AnimatedSprite rio = new(texture, animations)
    {
        CurrentAnimation = "walkdown",
        IsAnimating = true,
    };

    CharacterLayer chars = new();

    chars.Characters.Add(
        new Villager(rio, new(10, 10))
        {
            Position = new(320, 320),
            Tile = new(10, 10),
            Visible= true,
            Enabled=true,
            Conversation="Rio"
        });
```

```
    map.AddLayer(chars);

    EncounterLayer encounters = new();

    Encounter encounter = new(Player);
    encounter.Enemies.Add(Mob.FromString("Name=Giant Bat,Strength=3,Agil-
ity=5,Health=21,Position=640:640,Tile=5:5,AnimatedSprite=32x32-bat-
sprite;down:0:0:32:32:4;right:0:32:32:32:4;up:0:64:32:32:4;left:0:96:32:32:4;down",
Game.Content));

    encounters.Encounters.Add(((Mob)encounter.Enemies[0]).AnimatedSprite, encounter);
    map.AddLayer(encounters);

    rightButton = new(Game.Content.Load<Texture2D>("GUI/g21245"), ButtonRole.Menu)
    {
        Position = new(80, Settings.BaseHeight - 80),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    rightButton.Down += RightButton_Down;
    ControlManager.Add(rightButton);

    upButton = new(Game.Content.Load<Texture2D>("GUI/g21263"), ButtonRole.Menu)
    {
        Position = new(48, Settings.BaseHeight - 48 - 64),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    upButton.Down += UpButton_Down;
    ControlManager.Add(upButton);

    downButton = new(Game.Content.Load<Texture2D>("GUI/g21272"), ButtonRole.Menu)
    {
        Position = new(48, Settings.BaseHeight - 48),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    downButton.Down += DownButton_Down;
    ControlManager.Add(downButton);

    leftButton = new(Game.Content.Load<Texture2D>("GUI/g22987"), ButtonRole.Menu)
    {
        Position = new(16, Settings.BaseHeight - 80),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    leftButton.Down += LeftButton_Down;

    ControlManager.Add(leftButton);
}
```

The new code starts after creating the character layer and adding it to the map. First, create a new

encounter layer. Now, create a new encounter passing in the Player. Now, add an enemy to the list of enemies on the layer. Since it implements ICharacter, I can pass in a mob. I pass in some dummy values, for testing purposes. The important ones are that I pass in an animated sprite and a tile. After creating the encounter, I pass in the animated sprite. I need to cast it to a Mob, even though it is one. I then add the layer to the list of layers for the map.

If you build and run now, you will see the bat flapping drunkenly at tile (5, 5). I know it is a shorter tutorial that others, but I'm going to park this tutorial here. There is more that I could pack in, but I will save it for the following tutorials. I don't want you to have too much to digest at once. I encourage you to visit the news page of my site, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials. Also, I'm thinking of reviving my newsletter so you will be informed of new stuff rather than having to keep looking for new things.

Good luck with your game programming adventures!
Cynthia