Eyes of the Dragon Tutorials
Part 45
Mobs

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, I was going to start with the battle engine. However, I rapidly found that I was missing mobs, monsters, creatures, enemies, etc for the player to fight. I will call them mobs going forward. In this tutorial I will be adding in mobs to the game. So, let's get started!

To begin, right click the MGRpgLibrary project, select Add and then 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 starting code for that class.

```
using MGRpgLibrary.ItemClasses;
using MGRpgLibrary.SpriteClasses;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RpgLibrary.CharacterClasses;
using System;
using System.Collections.Generic;
using System.Text;

namespace MGRpgLibrary.Mobs
{
    public abstract class Mob
    {
        #region Constant Region

        public const int AttackRadius = 32;

        #endregion

        #region Field Region

        protected Entity entity;
        protected AnimatedSprite sprite;

        #endregion

        #region Property Region

        public Entity Entity
        {
            get { return entity; }
        }

        public AnimatedSprite Sprite
        {
            get { return sprite; }
```

```
        }

        #endregion

        #region Money Region

        private int _gold;

        public int Gold { get => _gold; }

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

        #endregion

        #region Item Drop Region

        protected List<GameItem> _drops = new List<GameItem>();

        public List<GameItem> Drops { get => _drops; }

        #endregion

        #region Constructor Region

        public Mob(Entity entity, AnimatedSprite sprite)
        {
            this.entity = entity;
            this.sprite = sprite;
        }

        #endregion

        #region Method Region
        #endregion

        #region Virtual Method region

        public virtual void Update(GameTime gameTime)
        {
            entity.Update(gameTime.ElapsedGameTime);
            sprite.Update(gameTime);
        }

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

        #endregion

        #region Abstract Method Region

        public abstract void Attack(Entity target);

        #endregion
    }
}
```

There are a bunch of using statements to bring classes into scope. This is an abstract class that must be inherited from. I went this route rather than adding mobs to the editor, for simplicity sake. Also, I feel that you've had more than enough of editors and would rather deal with code. There is a constant, AttackRadius, that is similar to the SpeakingRadius constant from the Character class. Instead of the player activate it, though, it will be triggered if the player gets that close to the mob. There are four protected fields with fields to expose their values. The first is protected and is an Entity, which is the entity associated with the mob. The next is an AninatedSprite, which is the sprite for the mob. The next is an integer and is the amount of gold the mob drops when defeated. The last is a List<GameItem> that is loot the mob could drop when defeated.

The constructor takes two parameters: an Enity and an AnimatedSprite. It sets the corresponding fields to the values passed in.

There are two virtual methods: Update and Draw. The Update method is the same as the Update method from the Character class. It takes a GameTime parameter and calls the Update method of the AnimatedSprite. The Draw method takes a GameTime parameter and a SpriteBatch parameter. It calls the Draw method of the AnimatedSprite. There is also an abstract method, Attack, that does an attack against an Entity.

Now I am going to add a couple mobs. The first is a bandit. Right click the Mobs folder, select Add and then Class. Name this new class Bandit. Here is the code for that class.

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

namespace MGRpgLibrary.Mobs
{
    public class Bandit : Mob
    {
        public Bandit(Entity entity, AnimatedSprite sprite)
            : base(entity, sprite)
        {
        }

        public override void Attack(Entity target)
        {
            if (Mechanics.RollDie(DieType.D20) >= 10 + Mechanics.GetModifier(target.Dexterity))
            {
                target.ApplyDamage(entity.MainHand);
            }
        }
    }
}
```

This is a fairly straight forward mob. It only implements the Attack method, that will be used in the battle engine. I went ahead and filled in the attack method. In an if statement I call the RollDie method of the Mechanics class passing in a D20. If that value is greater than or equal to 10 plus the dexterity modifier for the dexterity of the target the attack was successful and the ApplyDamage method is

called on the target.

There are two new methods, one in the Entity class and one in the Mechanics class. I will first add the GetModifier method of the Mechanics class. Add the following method to the Mechanics class.

```
public static int GetModifier(int value)
{
    if (value <= 3)
    {
        return -4;
    }

    if (value <= 5)
    {
        return -2;
    }

    if (value <= 8)
    {
        return -1;
    }

    if (value <= 12)
    {
        return 0;
    }

    if (value <= 14)
    {
        return 1;
    }

    if (value <= 16)
    {
        return 2;
    }

    if (value <= 18)
    {
        return 3;
    }

    if (value <= 20)
    {
        return 4;
    }

    return 5;
}
```

This is a pretty straight forward method. If the value passed in is less than or equal to three I return -4. Otherwise, if the value is less than or equal to 5 I return -2, if it is less than 9 I return -1, less than 12 returns 0, less than 14 returns 1, less than 17 returns 2, less than 19 returns 3, less than 21 returns 4, and finally returns 5 if greater than 20. You are encouraged to tweak these values to fit your game.

Now I will add the ApplyDamage method to the Entity class. I will add that this method is only for

physical attacks. Magic attacks will use a different method. Add this method to the Entity class.

```
public void ApplyDamage(GameItem mainHand)
{
    int defense = 0;

    if (offHand != null && offHand.Item is Shield shield)
    {
        defense += shield.DefenseValue;
    }

    if (head != null && head.Item is Armor armor)
    {
        defense += armor.DefenseValue;
    }

    if (body != null && body.Item is Armor bodyArmor)
    {
        defense += bodyArmor.DefenseValue;
    }

    if (hands != null && hands.Item is Armor handArmor)
    {
        defense += handArmor.DefenseValue;
    }

    if (feet != null && feet.Item is Armor footArmor)
    {
        defense += footArmor.DefenseValue;
    }

    int attack = 0;

    if (mainHand != null && mainHand.Item is Weapon weapon)
    {
        attack = weapon.AttackValue + weapon.AttackModifier;

        if (attack > defense)
        {
            ushort damage = 0;

            foreach (DamageEffectData e in weapon.DamageEffects)
            {
                for (int i = 0; i < e.NumberOfDice; i++)
                {
                    damage += (ushort)Mechanics.RollDie(e.DieType);
                }
            }

            Health.Damage(damage);
        }
    }
}
```

The method takes a GameItem parameter that is the weapon that is hitting the target. It first introduces a local variable defense that is the defense value of the character's armor. To find the value I check to see if there is an item in the off hand and it is shield. If there is a shield in the off hand its defense value is added to the defense variable. Similarly, if there is head armor I get its defense value

and add it to the variable. The same is true for the body, feet and hands. I then introduce a local variable, attack, that will hold the attack value of the weapon. If the mob is holding a weapon I assign the local variable to be the attack value of the weapon plus the attack modifier. If the attack is greater than the defense the weapon does damage to the entity. I loop over the DamageEffectData collection of the item and calculate the damage to be applied. Finally, I call the Damage method of the Health property to damage the entity.

Looking at this I noticed something. A character fully outfitted in chain mail would not be damaged by a long sword attack using this method. That is clearly wrong and the values for the armor and weapons will need to be adjusted! I will handle that in a future tutorial.

That is all the mobs that I am going to add in this tutorial. I will add more in a future tutorial. What I am going to do now is extend the TileMap so that there is a mob layer that will hold the mobs. Right click the TileEngine folder, select Add and then Class. Name this new class MobLayer. Here is the code for the MobLayer class.

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

namespace MGRpgLibrary.TileEngine
{
    public class MobLayer : ILayer
    {
        private Dictionary<Rectangle, Mob> _mobs = new Dictionary<Rectangle, Mob>();

        public Dictionary<Rectangle, Mob> Mobs { get => _mobs; }

        public void Draw(SpriteBatch spriteBatch, Camera camera, List<Tileset> tilesets)
        {
            foreach (Rectangle r in _mobs.Keys)
            {
                _mobs[r].Sprite.Position = new Vector2(r.X, r.Y);
                _mobs[r].Draw(new GameTime(), spriteBatch);
            }
        }

        public void Update(GameTime gameTime)
        {
            foreach (Mob m in _mobs.Values)
            {
                m.Update(gameTime);
            }
        }
    }
}
```

This class is very similar to the CharacterLayer class. It implements the ILayer interface so it can be added to the list of layers and have its draw and update methods called automatically. It has a single field that is a Dictionary<Rectangle, Mob> that holds the mobs in the layer. There is a get only accessor to expose the value of the field. The Draw method loops over the rectangles in the keys of

the dictionary. It sets the position of the sprite to the X and Y coordinates of the rectangle. It then calls the Draw method of the mob. The Update method just calls the Update method of the mob.

I then added a data class to hold the mobs and be read fro disk when run. Add a MobLayerData class to the TileEngine folder. Here is the code.

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

namespace MGRpgLibrary.TileEngine
{
    public class MobLayerData
    {
        public Dictionary<Rectangle, Mob> Mobs = new Dictionary<Rectangle, Mob>();
    }
}
```

Just a basic class with a single field, like the CharacterLayerData class. It is meant for serializing and deserializing the layer.

The next thing to do is add a mob to the map. I will be doing that in the LoadWorld method of the CharacterGeneratorScreen class. Update that method as follows.

```
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();
    MobLayer mobLayer = new MobLayer();

    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);
    map.AddLayer(mobLayer);

    Level level = new Level(map);

    ChestData chestData = Game.Content.Load<ChestData>(@"Game\Chests\Plain Chest");
```

```
    Chest chest = new Chest(chestData);

    BaseSprite chestSprite = new BaseSprite(
        containers,
        new Rectangle(0, 0, 32, 32),
        new Point(10, 10));

    ItemSprite itemSprite = new ItemSprite(
        chest,
        chestSprite);

    level.Chests.Add(itemSprite);

    World world = new World(GameRef, GameRef.ScreenRectangle);

    world.Levels.Add(level);
    world.CurrentLevel = 0;

    AnimatedSprite s = new AnimatedSprite(
        GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
        AnimationManager.Instance.Animations);

    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"), "FullSheet", new
Rectangle(1696, 1408, 32, 32)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetWeapon("Short Sword"), "FullSheet", new
Rectangle(800, 1504, 32, 32)));

    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;

    ed = new EntityData("Bandit", 1, 10, 12, 12, 10, 10, 10, "20|CON|10", "12|WIL|12", "0|0|
0");
```

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

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

    Mob mob = new Bandit(e, s);
    ((MobLayer)world.Levels[world.CurrentLevel].Map.Layers.Find(x => x is
MobLayer)).Mobs.Add(new Rectangle(0, 512, 32, 32), mob);
    mob.Entity.Equip(new GameItem(ItemManager.GetWeapon("Short Sword"), "FullSheet", new
Rectangle(800, 1504, 32, 32)));

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

What is new here is I create a MobLayer and add it to the layers on the map. Later on I create an EntityData for the bandit. I then create an Enity using the EntityData. I then create an AnimatedSprite using the image for the male rogue from the player's sprite. I didn't want to search for an image that fit the pattern. I then create a Bandit object and assign it to a Mob variable. Technically you could just use a Bandit variable but I like using the base class. Then like with adding a merchant to the map I add the mob to the map. Finally I equip a short sword on the bandit.

If you run and build now you will see the bandit down the map on the right hand side of the game. I'm going to save detection to another tutorial. So, that is going to be it for this tutorial. I accomplished what I intended, have a class for mobs and being able to add them to the map. 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