Eyes of the Dragon Tutorials
Part 34
Non-Player Character Sprites

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, the game is coming along well but the player cannot interact with NPCs because they are none on the map. In this tutorial I will add an NPC to the map and attach a conversation to them.

The first thing that I want to do is add a class that will be a repository for all of the animations in the game. This will reduce duplicated code in several modules and give a single point of use. In theory it also allows for separation of duties. This is because now that it is centralized one developer will "own" this module and any other developers working on the game interact with it using the publicly defined contract. They do not need to know anything about how it has been implemented.

Right click the SpriteClasses folder in the MGRpgLibrary project, select Add and then Class. Name this new class AnimationManager. This is the code for that class.

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

namespace MGRpgLibrary.SpriteClasses
{
    public sealed class AnimationManager
    {
        private static readonly AnimationManager instance = new AnimationManager();
        private readonly Dictionary<AnimationKey, Animation> animations = new
            Dictionary<AnimationKey, Animation>();

        public static AnimationManager Instance
        {
            get { return instance; }
        }

        public Dictionary<AnimationKey, Animation> Animations
        {
            get { return animations; }
        }

        private AnimationManager()
        {
            Animation animation = new Animation(3, 32, 32, 0, 0);
            animations.Add(AnimationKey.Down, animation);

            animation = new Animation(3, 32, 32, 0, 32);
            animations.Add(AnimationKey.Left, animation);
```

```
            animation = new Animation(3, 32, 32, 0, 64);
            animations.Add(AnimationKey.Right, animation);

            animation = new Animation(3, 32, 32, 0, 96);
            animations.Add(AnimationKey.Up, animation);
        }
    }
}
```

This class was created with the Singleton Design Pattern because there should only ever be one instance of this class at any given time. For that reason I sealed the class so it cannot be inherited from. There is then a static field that generates an instance using a private constructor. I also included a field that is of type Dictionary<AnimationKey, Animation> the same way that I did in the character creation screen when I created the player. Next up is a static property that exposes the instance to outside classes and another property for the dictionary of animations. The constructor just creates the animations as was done previously when creating the player character.

The next few changes will be to the CharacterGeneratorScreen class. First, I will refactor the CreatePlayer method to use the new animation manager that we created instead of creating animations manually. Please update that method with the following code.

```
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);
}
```

The next step is to generate an NPC for the player to talk with and add them to the level. For that we will need a sprite sheet for the NPC. You can download the sprite sheet that I created with this link: https://cynthiamcmahon.ca/blog/downloads/Eliza.png. After you have downloaded the sprite sheet open the MonoGame Pipeline Tool. Right click the Content node, select Add and then New Folder. Name this new folder SpriteSheets. Right click the SpriteSheets folder, select Add and then Existing Item. Navigate to the Eliza.png file and add it to the project. Save the project and close the tool.

Next up is to create the character on the fly and add it to the map. Replace the CreateWorld method in the CharacterGeneratorScreen with the following code.

```
private void CreateWorld()
{
    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
    Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");

    Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);
    MapLayer layer = new MapLayer(100, 100);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);
            layer.SetTile(x, y, tile);
        }
    }

    MapLayer splatter = new MapLayer(100, 100);
    Random random = new Random();

    for (int i = 0; i < 100; i++)
    {
        int x = random.Next(0, 100);
        int y = random.Next(0, 100);
        int index = random.Next(2, 14);

        Tile tile = new Tile(index, 0);

        splatter.SetTile(x, y, tile);
    }

    splatter.SetTile(1, 0, new Tile(0, 1));
    splatter.SetTile(2, 0, new Tile(2, 1));
    splatter.SetTile(3, 0, new Tile(0, 1));

    TileMap map = new TileMap(tileset1, layer);

    map.AddTileset(tileset2);
    map.AddLayer(splatter);
    map.CollisionLayer.Collisions.Add(new Point(1, 0), CollisionType.Impassable);
    map.CollisionLayer.Collisions.Add(new Point(3, 0), CollisionType.Impassable);
```

```
    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(5 * Engine.TileWidth, 5 * Engine.TileHeight);

    EntityData ed = new EntityData("Eliza", 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);

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

    GamePlayScreen.World = world;
}
```

First, I create a new animated sprite using the sprite sheet that was just added to the project. I use the AnimationManager to generated the animations. Next up is setting the position of the sprite to (5, 5) on the map. Next I create an EntityData object that represents the NPC. It would have been better to use one of the character classes that have already been defined but this works well for now. Once the entity data has been created I can create an entity that represents the NPC. Finally with an entity and sprite I can create an NPC add add her to the current level.

If you build and run the game now you will see the new character sitting happily doing nothing on the screen. If you move the player to her position though you just walk right over top of her. The next step will be to add some collision detection so that the player cannot walk over other characters on the map.

First, open up the Character class in the MGRpgLibrary project. Now add the following region to the class that will be used for collision detection and checking to see if the character is close enough for the player to start a conversation with them.

```
        #region Constant Region
```

```
        public const int SpeakingRadius = 32;
        public const int CollisionRadius = 8;

        #endregion
```

Implementing collision detection will be done in the Player component so open that class up. You can update the UpdatePosition method to the following code.

```
private void UpdatePosition(GameTime gameTime, Vector2 motion)
{
    Sprite.IsAnimating = true;
    motion.Normalize();

    Vector2 distance = motion * Sprite.Speed *
        (float)gameTime.ElapsedGameTime.TotalSeconds;
    Vector2 next = distance + Sprite.Position;

    Rectangle playerRect = new Rectangle(
        (int)next.X + CollisionLayer.CollisionRadius,
        (int)next.Y + CollisionLayer.CollisionRadius,
        Engine.TileWidth - CollisionLayer.CollisionRadius,
        Engine.TileHeight - CollisionLayer.CollisionRadius);
    foreach (Point p in
        GamePlayScreen.World.CurrentMap.CollisionLayer.Collisions.Keys)
    {
        Rectangle r = new Rectangle(
            p.X * Engine.TileWidth + CollisionLayer.CollisionRadius,
            p.Y * Engine.TileHeight + CollisionLayer.CollisionRadius,
            Engine.TileWidth - CollisionLayer.CollisionRadius,
            Engine.TileHeight - CollisionLayer.CollisionRadius);

        if (r.Intersects(playerRect))
        {
            return;
        }
    }

    foreach (Character c in
        GamePlayScreen.World.Levels[GamePlayScreen.World.CurrentLevel].Characters)
    {
        Rectangle r = new Rectangle(
            (int)c.Sprite.Position.X + Character.CollisionRadius,
            (int)c.Sprite.Position.Y + Character.CollisionRadius,
            Engine.TileWidth - Character.CollisionRadius,
            Engine.TileHeight - Character.CollisionRadius);

        if (r.Intersects(playerRect))
        {
            return;
        }
    }

    Sprite.Position = next;
    Sprite.LockToMap();

    if (camera.CameraMode == CameraMode.Follow)
    {
        camera.LockToSprite(Sprite);
    }
}
```

Just like when checking for tile collisions I iterate through the characters that have been added to the level. I then create a rectangle that represents their bounds. Like for tile collision I remove a bit of padding from the sprite's bounds so that collision detection doesn't appear to be so rigid. If there is a

collision I return out of the method.
If you were to build and run the game now the player will be able to walk around the map still but not through the NPC that we just added. It would be nice to be able to start a conversation with her though. First, we need to set up a mechanism for the player to try to do that. Before I get to that I want to make an update to the AnimatedSprite class that will return the center pixel of the sprite on the screen. Add the following property to the AnimatedSprite class.

```
public Vector2 Center
{
    get { return new Vector2(Position.X + Width / 2, Position.Y + Height / 2); }
}
```

This property returns a new Vector2 object that is the sprite's position plus half its with and its height, giving us the center. Why exactly is that important here? Let's look at Maria here wanting to speak with Juliette.

They should be close enough for them to talk, right? How do you know that though? We could look at using their rectangles but they we would have to add padding to them like we are doing with our tile collisions. We don't have to do that though. Instead, we are going to draw a circle around each of the characters. If the circles over lap then they can talk with each other.

First, lets say that the circle is the height and width of the character, 16 pixels, based off the centers. That gives us the following scenario. In order to talk they would have to be right beside each other. They should be close enough to speak but they can't quite make it. That is why if you look at the SpeakingRadius that I added to the Character class it is 32 and not 16. Let's see what that gives us.

Now the game sees them as close enough to talk to one another and will start the conversation. This is really just another type of collision detection. Instead of using rectangles we are using circles to check to see if the two objects collide with each other. If they do collide and the player asks to start a conversation we will check to see if there is a conversation and if there is start one. In order to start the conversation we need to add that to our game screens/states. Update the Game1 class with the following code.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MGRpgLibrary;
using EyesOfTheDragon.Components;
using EyesOfTheDragon.GameScreens;

namespace EyesOfTheDragon
{
    public class Game1 : Game
    {
        #region Frames Per Second Field Region

        private float fps;
        private float updateInterval = 1.0f;
        private float timeSinceLastUpdate = 0.0f;
        private float frameCount = 0;

        #endregion

        private GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        private GameStateManager _gameStateManager;

        public const int ScreenWidth = 1280;
        public const int ScreenHeight = 720;

        public readonly Rectangle ScreenRectangle = new Rectangle(
            0,
            0,
            ScreenWidth,
            ScreenHeight);

        public SpriteBatch SpriteBatch { get { return _spriteBatch; } }
        public TitleScreen TitleScreen { get; private set; }
        public StartMenuScreen StartMenuScreen { get; private set; }
        public GamePlayScreen GamePlayScreen { get; private set; }
        public CharacterGeneratorScreen CharacterGeneratorScreen { get; private set; }
        public SkillScreen SkillScreen { get; private set; }
        public LoadGameScreen LoadGameScreen { get; private set; }

        public ConversationScreen ConversationScreen { get; private set; }

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

```
            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);

            _gameStateManager.ChangeState(TitleScreen);

            IsFixedTimeStep = false;
            _graphics.SynchronizeWithVerticalRetrace = false;
        }

        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            _graphics.PreferredBackBufferWidth = ScreenWidth;
            _graphics.PreferredBackBufferHeight = ScreenHeight;
            _graphics.ApplyChanges();

            base.Initialize();
        }

        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);
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            base.Draw(gameTime);
```

```
            float elapsed = (float)gameTime.ElapsedGameTime.TotalSeconds;
            frameCount++;
            timeSinceLastUpdate += elapsed;

            if (timeSinceLastUpdate > updateInterval)
            {
                fps = frameCount / timeSinceLastUpdate;

                System.Diagnostics.Debug.WriteLine("FPS: " + fps.ToString());
                this.Window.Title = "FPS: " + fps.ToString();

                frameCount = 0;
                timeSinceLastUpdate -= updateInterval;
            }
        }
    }
}
```

All that I did was add a property that will contain the conversation screen and create a new screen like I did with the other screens. Let's bring it all together and switch to a conversation screen from the game screen. First, I want to add two fields and properties to the NonPlayerCharacter class. They hold the current conversation associated with and the current quest associated with the character. Change the code for that class to the following.

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

namespace MGRpgLibrary.CharacterClasses
{
    public class NonPlayerCharacter : Character
    {
        #region Field Region

        readonly List<Conversation> conversations;
        readonly List<Quest> quests;
        private string currentConversation;
        private string currentQuest;

        #endregion

        #region Property Region

        public List<Conversation> Conversations
        {
            get { return conversations; }
        }

        public List<Quest> Quests
        {
            get { return quests; }
```

```
        }

        public string CurrentConversation
        {
            get { return currentConversation; }
        }

        public string CurrentQuest
        {
            get { return currentQuest; }
        }

        public bool HasConversation
        {
            get { return (conversations.Count > 0); }
        }

        public bool HasQuest
        {
            get { return (quests.Count > 0); }
        }

        #endregion

        #region Constructor Region

        public NonPlayerCharacter(Entity entity, AnimatedSprite sprite)
            : base(entity, sprite)
        {
            conversations = new List<Conversation>();
            quests = new List<Quest>();
        }

        #endregion

        #region Method Region
        #endregion

        #region Virtual Method region

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

        public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            base.Draw(gameTime, spriteBatch);
        }
        #endregion
    }
}
```

Let's bring that together into the game play screen. Change the Update method 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 (Character c in World.Levels[World.CurrentLevel].Characters)
        {
            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();
                }
            }
        }
    }

    base.Update(gameTime);
}
```

You also need to bring in the namespace for characters and NPCs. Add the following using statement to the GamePlayScreen.

```
using MGRpgLibrary.CharacterClasses;
```

In order to start a conversation I check to see if the space key or the A button on the game pad has been released. If it has I iterate over the list of characters in the level. I get the distance between the two centers using a static method of the Vector2 class. It does have a lot of really useful helper methods like this. I then check to see if the distance is less than the speaking radius that was defined earlier and that the character is a NonPlayerCharacter. If it is I cast the loop variable to an NonPlayerCharacter variable. Then I check to see if that character has a conversation assigned to them. If they do I push the conversation state on top of the stack, set the conversation to the current conversation and start the conversation.

If you add a breakpoint inside of the if statement that checks for the key release you can step through the code and see that it gets as far as the if statement that checks to see if the NPC has a conversation associated with them. Since they do not have a conversation associated with them there is no way to start it.

I'm not going to cover adding a conversation to the NPC in this tutorial as I find it a rather complicated topic and deserving of a tutorial of its own. Instead I'm going to wrap up the tutorial here because I'd like to try and keep the tutorials to a reasonable length so that you don't have too much to digest at

once. So, I encourage you to visit my blog at, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck in your Game Programming Adventures!

Cynthia