Eyes of the Dragon Tutorials
Part 46
Combat Engine – Part One

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.

In this tutorial I will be adding the long anticipated combat engine. The way combat is going to work is each round the player will be presented options to either attack, use a spell or talent, or run away. If the talent or spell is selected a pop up menu will come up with the player's spells/talents. I will be building it in stages to try and prevent walls of code. To start, right click the GameScreens folder, select Add and then Class. Name this new class CombatScreen. Here is the code for that class.

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

namespace EyesOfTheDragon.GameScreens
{
    public class CombatScreen : BaseGameState
    {
        private Texture2D _menuTexture;
        private Texture2D _actionTexture;
        private bool _displayActionTexture = false;
        private Mob _mob;

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

        public void SetMob(Mob mob)
        {
            _mob = mob;
        }

        protected override void LoadContent()
        {
            Color[] data = new Color[1280 * 200];

            for (int i = 0; i < data.Length; i++)
            {
                data[i] = Color.Black;
            }

            _menuTexture = new Texture2D(GraphicsDevice, 1280, 200);
            _menuTexture.SetData(data);

            data = new Color[200 * 720];
            for (int i = 0; i < data.Length; i++)
            {
                data[i] = Color.AliceBlue;
            }

            _actionTexture = new Texture2D(GraphicsDevice, 200, 720);
            _actionTexture.SetData(data);

            base.LoadContent();
        }

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

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

            GameRef.SpriteBatch.Begin();

            GameRef.SpriteBatch.Draw(_menuTexture, new Rectangle(0, 720 - _menuTexture.Height,
_menuTexture.Width, _menuTexture.Height), Color.White);

            if (_displayActionTexture)
            {
                GameRef.SpriteBatch.Draw(_actionTexture, new Rectangle(_actionTexture.Width, 0,
_actionTexture.Width, _actionTexture.Height), Color.White);
            }

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

So, there are some using statements to bring classes in hte MGRpgLibrary and MonoGame libraries into scope. Because this is a game state it inherits from BaseGameState. For fields there are two Texture2Ds, one for the menu that will display the options to attack, use a spell or skill, or run for your life! Technically, we want combat to be to the death. However, I'm giving the player the choice to run for their life. There is a bool field that when true will have the action texture displayed. Finally, there is a Mob field that is the mob being fought.

There is a public method, SetMob, that is used to set the mob being fought. Rather than create textures for the menu and action bar in Gimp I create them on the fly. I do that in the LoadContent method. What I do is create a Color array set the the width of the texture times the height of the texture. Next I loop over the elements of the array and assign them a color. I then create a texture using the constructor. Finally, I call the SetData method to assign the colors to the texture.

There is a stub for the Update method, for now. In the Draw method I draw the menu texture. If the action texture should be drawn I draw it as well.

The next step is to create the combat screen. That will be done in the Game1 class, like the other screens the we have created so far. Add the following property to the Game1 class above the constructor and update the constructor as well.

```
public CombatScreen CombatScreen { get; private set; }

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);

    _ = new TextureManager();

    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);
    InventoryScreen = new InventoryScreen(this, _gameStateManager);
    CombatScreen = new CombatScreen(this, _gameStateManager);

    _gameStateManager.ChangeState(TitleScreen);

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

I've already explained what the changes are, so I am just going to move along. The next change will be to update the GamePlayScreen to trigger a fight. I did the in the Update method. Change 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);
                    }
                }
            }
        }
    }

    MobLayer mobLayer = World.Levels[World.CurrentLevel].Map.Layers.Find(x => x is MobLayer) as
MobLayer;

    foreach (Rectangle r in mobLayer.Mobs.Keys)
    {
        float distance = Vector2.Distance(mobLayer.Mobs[r].Sprite.Center,
player.Sprite.Center);

        if (distance < Mob.AttackRadius)
        {
            GameRef.CombatScreen.SetMob(mobLayer.Mobs[r]);

            StateManager.PushState(GameRef.CombatScreen);
            Visible = true;
        }
    }

    if (InputHandler.KeyReleased(Keys.I))
    {
        StateManager.PushState(GameRef.InventoryScreen);
    }

    base.Update(gameTime);
}
```

What the new code does is call the Find method of the List<T> class to find the MobLayer. We should probably check for null but I am sure there will always be a mob layer on the map. Next I loop over the keys in the Dictionary that holds the mobs. I then calculate the distance between the center of the mob and the center of the player. If that distance is less then the attack radius for mobs I push the combat state on the stack of states. I call the SetMob method to set the mob and then set the Visible property of the game play state to true so it will be rendered.

If you build and run now, you can walk down to the bandit and trigger the combat state. So, piece one is done. We can trigger combat if we walk too close to a mob. The next step will be to display something on the combat state, other than the texture for the background. I will be extending the conversation classes to display a menu where the player will choose their course of action. Add the following using statement and field to the CombatScreen. Also, update the SetMob method in the CombatScreen to the following.

```
using MGRpgLibrary.ConversationComponents;

private GameScene _scene;

public void SetMob(Mob mob)
{
    _mob = mob;

    _scene = new CombatScene(Game, "Combat Scene", new List<SceneOption>());

    SceneOption option = new SceneOption("Attack", "Attack", new SceneAction());

    _scene.Options.Add(option);

    if (GamePlayScreen.Player.Character.Entity.Mana.MaximumValue > 0)
    {
        option = new SceneOption("Spell", "Spell", new SceneAction());
    }
    else
    {
        option = new SceneOption("Talent", "Talent", new SceneAction());
    }
    _scene.Options.Add(option);

    option = new SceneOption("Run for your life!", "Run for your life!", new SceneAction());
    _scene.Options.Add(option);

    ((CombatScene)_scene).SetCaption($"You face death itself in the form of a
{_mob.Entity.EntityName.ToLower()}.");
}
```

This should be familiar by now but I will go over it. I created a GameScene field that will display the options for the fight. In the Initialize method I create a CombatScene with the Game property of the component, set it to a basic_scene with an empty string and then a new List<SceneOption>. I then create a SceneOption with the values Attack, Attack, and a new scene action. The option is then added to the list of options for the scene. To determine if the player's character is a spell caster or not I check the MaximumMana property of the Mana property of the Entity property of the Character. Wow, that is really drilling down. If the player is a spell caster I create a SceneOption with Spell. Otherwise, I create a SceneOption with Talent. The option is then added to the list of scene options. Finally, I create an option for running and add it to the list of scene options. After creating the scene, I call a method SetCaption that displays some text at the top of the scene.

So, now we need the new CombatScene. It is close to the GameScene that it inherits from. The first step is to make the rest of the fields in the GameScene protected instead of private. Replace the fields of the GameScene class to the following.

```
protected Game game;
protected string textureName;
protected Texture2D texture;
protected SpriteFont font;
protected string text;
protected List<SceneOption> options;
protected int selectedIndex;
protected Color highLight;
protected Color normal;
protected Vector2 textPosition;
protected static Texture2D selected;
protected Vector2 menuPosition = new Vector2(50, 475);
```

The other change is to make the Update and Draw method of the GameScene class virtual so they can be overridden in the new class. Replace the Update and Draw method signatures to the following code.

```
public virtual void Update(GameTime gameTime, PlayerIndex index)
public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch, Texture2D portrait = null)
```

Now, we need to create the CombatScene class. Right click the ConversationComponents folder, select Add and then class. Name this new class CombatScene. Here is the code for that class.

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

namespace MGRpgLibrary.ConversationComponents
{
    public class CombatScene : GameScene
    {
        private string caption;

        public CombatScene(Game game, string text, List<SceneOption> options, string
textureName = "basic_scene") : base(text, options, textureName)
        {
            menuPosition = new Vector2(50, 530);
            NormalColor = Color.White;
            HighLightColor = Color.Red;
            this.game = game;
        }

        public void SetCaption(string caption)
        {
            this.caption = caption;
        }

        public override void Draw(GameTime gameTime, SpriteBatch spriteBatch, Texture2D
portrait = null)
        {
            Vector2 selectedPosition = new Vector2();
            Rectangle portraitRect = new Rectangle(25, 25, 425, 425);
            Color myColor;

            if (selected == null)
                selected = game.Content.Load<Texture2D>(@"GUI\rightarrowUp");

            if (textPosition == Vector2.Zero)
                SetText(text);

            if (portrait != null)
                spriteBatch.Draw(portrait, portraitRect, Color.White);

            font = FontManager.GetFont("testfont");

            Vector2 position = menuPosition;

            spriteBatch.DrawString(font, caption, menuPosition, Color.Yellow);

            position.Y += font.LineSpacing + 10;

            for (int i = 0; i < options.Count; i++)
            {
                if (i == SelectedIndex)
                {
                    myColor = HighLightColor;
                    selectedPosition.X = position.X - 35;
                    selectedPosition.Y = position.Y;
                    spriteBatch.Draw(selected, selectedPosition, Color.White);
                }
                else
                    myColor = NormalColor;

                spriteBatch.DrawString(font,
                    options[i].OptionText,
                    position,
                    myColor);

                position.Y += font.LineSpacing + 5;
            }
        }
    }
}
```

There are some using statements to bring MonoGame classes into scope. The class inherits from GameScene. There is a single field, caption, that will be drawn at the top of the scene. The constructor takes as parameters a Game object, the text for the scene, a List<SceneOption> and an optional texture name. It calls the base constructor that takes the text, List<SceneOption>, and a texture name. It initializes the position to draw the scene, the normal color, hightlight color, and sets the game field. The SetCaption method just sets the caption for the scene.

The override of the Draw method takes three parameters: a GameTime object, SpriteBatch object, and an optional parameter for the background to be drawn. It flows similarly to the GameScene class. I want to completely override the behavior so I don't call the base method. I create a local variable that is the position of the caret. I then create a rectangle that describes the position of the portrait, if any. There is then a local variable that holds the color to draw text in. If the selected texture is null I load it.

If the textPosition field is the zero vector I call the SetText method of the base class to position the text. If the portrait is not null I draw it. I then set the font field to the font from the font manager. I then create a local variable to determine where to draw text. I first draw the caption. I then increase the Y property of the position to the line spacing of the font plus a little padding. You may want to increase the padding in your game but 10 seems reasonable.

Next, I loop over the options for the scene. If the index is the current option I set the color to draw text in to the high light color. I then position the caret to the left of the text. I then draw the caret. Otherwise, I set the color to draw text in to the normal color. I then draw the text for the option in the appropriate color. I then update the position of the text to be the line spacing plus padding of 5 pixels.

In the Update method I need to call the Update method of the _scene field. Similarly, I need to call the Draw method of the _scene field in the Draw method. Update those methods to the following code.

```
public override void Update(GameTime gameTime)
{
    _scene.Update(gameTime, PlayerIndex.One);

    base.Update(gameTime);
}

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

    GameRef.SpriteBatch.Begin();

    GameRef.SpriteBatch.Draw(_menuTexture, new Rectangle(0, 720 - _menuTexture.Height,
_menuTexture.Width, _menuTexture.Height), Color.White);

    _scene.Draw(gameTime, GameRef.SpriteBatch);
    if (_displayActionTexture)
    {
        GameRef.SpriteBatch.Draw(_actionTexture, new Rectangle(_actionTexture.Width, 0,
_actionTexture.Width, _actionTexture.Height), Color.White);
    }

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

Nothing else special here, as far as the code goes. What I do need to do is update the Draw method of the GameScene class. What I need to do is make the third parameter optional, rather than always passing in null if there is no portrait. Replace the Draw method of the GameScene class to the following.

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Texture2D portrait = null)
{
    Vector2 selectedPosition = new Vector2();
    Rectangle portraitRect = new Rectangle(25, 25, 425, 425);
    Color myColor;

    if (selected == null)
        selected = game.Content.Load<Texture2D>(@"GUI\rightarrowUp");
    if (textPosition == Vector2.Zero)
        SetText(text);

    if (texture == null)
        texture = game.Content.Load<Texture2D>(@"GameScenes\" + textureName);

    spriteBatch.Draw(texture, Vector2.Zero, Color.White);

    if (portrait != null)
        spriteBatch.Draw(portrait, portraitRect, Color.White);

    spriteBatch.DrawString(font,
        text,
        textPosition,
        Color.White);

    Vector2 position = menuPosition;

    for (int i = 0; i < options.Count; i++)
    {
        if (i == SelectedIndex)
        {
            myColor = HighLightColor;
            selectedPosition.X = position.X - 35;
            selectedPosition.Y = position.Y;
            spriteBatch.Draw(selected, selectedPosition, Color.White);
        }
        else
            myColor = NormalColor;

        spriteBatch.DrawString(font,
            options[i].OptionText,
            position,
            myColor);

        position.Y += font.LineSpacing + 5;
    }
}
```

If you build and run now, you can bump into the mob and trigger the combat. You will see the caption along with a menu that allows for selecting an option. However, it doesn't do anything and you are stuck in the combat. Let's add some logic to the CombatScreen to attack and run. Change the Update method of the CombatScreen to the following.

```
        public override void Update(GameTime gameTime)
        {
            _scene.Update(gameTime, PlayerIndex.One);

            if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Space))
            {
                switch (_scene.SelectedIndex)
                {
                    case 0:
                        _mob.Attack(GamePlayScreen.Player.Character.Entity);
                        break;
                    case 2:
                        Vector2 center = GamePlayScreen.Player.Sprite.Center;
                        Vector2 mobCenter = _mob.Sprite.Center;
                        if (center.Y < mobCenter.Y)
                        {
                            GamePlayScreen.Player.Sprite.Position.Y -= 8;
                        }
                        else if (center.Y > mobCenter.Y)
                        {
                            GamePlayScreen.Player.Sprite.Position.Y += 8;
                        }
                        else if (center.X < mobCenter.X)
                        {
                            GamePlayScreen.Player.Sprite.Position.X -= 8;
                        }
                        else
                        {
                            GamePlayScreen.Player.Sprite.Position.X += 8;
                        }

                        StateManager.PopState();
                        break;
                }
            }
            base.Update(gameTime);
        }
```

What I did here was is check to see if the space key has been released. If it has there is a switch on the SelectedIndex property of the scene. If the selected index is 0, I call the Attack method on the mob passing in the player's entity. Here is where the mob should hit back but I'm not getting into that in this tutorial. Then if the selected index is 1 I would display the action menu but I am not getting into that in this tutorial. Finally, if the selected index is 2 I grab the center of the player's sprite and the center of the mob's sprite. I then check to see if the Y component of the player's center is less than the Y component of the mob's center. If it is I subtract 8 pixels from the player's sprite's position. The reason why I do this is without altering the player's position they would be thrown right back into the fight. Then, if the player's Y component of its center is greater than the Y component of the mob's center I add 8 pixels to the player's sprite's Y component. I then do something similar for the X component of the centers.

If you run and build now you can initiate combat with the bandit and run away. You can hit the bandit but the bandit does not hit back and never dies. I will fix these items, and more, in the next tutorial. So, that is going to be it for this tutorial. I accomplished what I intended, we have the mechanics for starting a fight. I don't want to venture further in this tutorial. So, please continue to 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