Eyes of the Dragon Tutorials
Part 49
Combat Engine – Part Four

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 continuing on with the combat engine. I will also add more game items to the merchant for the player to buy. So, let's get started.

First, I am going to make a few updates to the combat engine. I am going to add some code to the DoAction method. It will also fix a bug that I found where if you use a talent or spell you are stuck selecting spells or talents. Change the DoAction method of the CombatScreen class to the following.

```
private void DoAction()
{
    Entity entity = GamePlayScreen.Player.Character.Entity;

    if (entity.Mana.MaximumValue > 0)
    {
        SpellData spell = DataManager.SpellData.SpellData[_actions[_action]];

        if (spell.ActivationCost > entity.Mana.CurrentValue)
        {
            return;
        }

        entity.Mana.Damage((ushort)spell.ActivationCost);

        Spell s = Spell.FromSpellData(spell);

        if (s.SpellType == SpellType.Activated)
        {
            foreach (BaseEffect e in s.Effects)
            {
                switch (e.TargetType)
                {
                    case TargetType.Enemy:
                        e.Apply(_mob.Entity);
                        break;
                    case TargetType.Self:
                        e.Apply(entity);
                        break;
                }
            }
        }
    }
    else
    {
        TalentData talent = DataManager.TalentData.TalentData[_actions[_action]];

        if (talent.ActivationCost > entity.Stamina.CurrentValue)
        {
            return;
        }

        entity.Stamina.Damage((ushort)talent.ActivationCost);

        Talent s = Talent.FromTalentData(talent);

        if (s.TalentType == TalentType.Activated)
        {
            foreach (BaseEffect e in s.Effects)
            {
                switch (e.TargetType)
                {
                    case TargetType.Enemy:
                        e.Apply(_mob.Entity);
                        break;
                    case TargetType.Self:
                        e.Apply(entity);
                        break;
                }
            }
        }
    }

    _displayActionTexture = false;
    _mob.DoAttack(entity);
}
```

All I did was set _displayActionTexture to false after doing the action. The next thing I'm going to do is add some visual cues as to what the status of the player and the enemy is. I will do that by adding another texture to the screen that will hold the player's data and the enemy's data. First, add these new fields and modify the LoadContent method to the following.

```
private Texture2D _hudTexture;
private Texture2D _healthBorderTexture;
private Texture2D _healthTexture;

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.Blue;
    }

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

    data = new Color[1280 * 40];

    for (int i = 0; i < 1280 * 40; i++)
    {
        data[i] = Color.Black;
    }

    _hudTexture = new Texture2D(GraphicsDevice, 1280, 40);
    _hudTexture.SetData(data);

    data = new Color[502 * 12];

    for (int i = 0; i < 502 * 12; i++)
    {
        data[i] = Color.Gray;
    }

    _healthBorderTexture = new Texture2D(GraphicsDevice, 502, 12);
    _healthBorderTexture.SetData(data);

    data = new Color[500 * 10];

    for (int i = 0; i < 500 * 10; i++)
    {
        data[i] = Color.White;
    }

    _healthTexture = new Texture2D(GraphicsDevice, 500, 10);
    _healthTexture.SetData(data);

    base.LoadContent();
}
```

Nothing really new here. It is just creating some textures for the combat engine rather than drawing them in a paint program and loading them through the content manager. There is a texture that is the background for the HUD, a texture for a border around health and stamina/mana bars. Finally there is a texture for the bars. There is just one bar that we will tint red for health, blue for stamina and green for mana.

Now that we have the textures it is time to render them. That is, of course, done in the Draw method. Replace the Draw method of the CombatScreen with the following.

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

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

    GameRef.SpriteBatch.Begin();

    Rectangle menuDest = new Rectangle(0, 720 - _menuTexture.Height, _menuTexture.Width,
_menuTexture.Height);
    Rectangle hudDest = new Rectangle(0, menuDest.Top - _hudTexture.Height, _hudTexture.Width,
_hudTexture.Height);
```

```
    GameRef.SpriteBatch.Draw(_menuTexture, menuDest, Color.White);
    GameRef.SpriteBatch.Draw(_hudTexture, hudDest, Color.White);

    Rectangle dest = new Rectangle(10, hudDest.Top + 12, _healthBorderTexture.Width,
_healthBorderTexture.Height);
    GameRef.SpriteBatch.Draw(_healthBorderTexture, dest, Color.White);

    dest.X += 5;
    dest.Y += 2;
    dest.Height -= 5;
    dest.Width = (int)(dest.Width *
((float)GamePlayScreen.Player.Character.Entity.Health.CurrentValue /
GamePlayScreen.Player.Character.Entity.Health.MaximumValue)) - 10;

    GameRef.SpriteBatch.Draw(_healthTexture, dest, Color.Red);

    dest = new Rectangle(Game1.ScreenWidth - _healthBorderTexture.Width - 10, hudDest.Top + 12,
_healthBorderTexture.Width, _healthBorderTexture.Height);
    GameRef.SpriteBatch.Draw(_healthBorderTexture, dest, Color.White);

    dest.X += 5;
    dest.Y += 2;
    dest.Height -= 5;
    dest.Width = (int)(dest.Width * ((float)_mob.Entity.Health.CurrentValue /
_mob.Entity.Health.MaximumValue)) - 10;

    GameRef.SpriteBatch.Draw(_healthTexture, dest, Color.Red);

    _scene.Draw(gameTime, GameRef.SpriteBatch);

    if (_displayActionTexture)
    {
        GameRef.SpriteBatch.Draw(_actionTexture, new Rectangle(_actionTexture.Width, 0,
_actionTexture.Width, _actionTexture.Height), Color.White);
        Vector2 position = new Vector2(_actionTexture.Width + 50, 5);

        int count = 0;

        foreach (string s in _actions)
        {
            Color myColor = Color.Black;

            if (count == _action)
            {
                myColor = Color.White;
                GameRef.SpriteBatch.Draw(_selected, new Rectangle((int)position.X - 35,
(int)position.Y, _selected.Width, _selected.Height), Color.White);
            }

            GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"), s, position,
myColor);
            position.Y += FontManager.GetFont("testfont").LineSpacing + 5;

            count++;
        }
    }

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

What has changed is I calculate where to draw heads up display texture. That was done by taking the Top property of the menu and subtracting the height of the texture. The heads up display is the full width of the window so the X properties are 0 and the width of the texture, and the height is the height of the texture. The next change is I render it.

After rendering the texture, I create a destination rectangle for the player's health bar border. I position it 10 pixels to the right of the edge and 12 pixels down from the top of the heads up display. Its width is the width of the texture and its height is the height of the texture. I then draw the texture. I then make some adjustments to the destination rectangle. They shrink the destination so that the health bar is more or less centered in the border texture, except for the width. The width is a percentage of the current health divided by the maximum health. I then render the health texture. I repeat the process for the mob's health.

I need to update the TalentDataManager class. I didn't set the target when creating the talents. I just added an additional field when initializing the objects. Replace the TalentDataManager class with the following code.

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

namespace RpgLibrary.TalentClasses
{
    public class TalentDataManager
    {
        #region Field Region

        readonly Dictionary<string, TalentData> talentData;

        #endregion

        #region Property Region

        public Dictionary<string, TalentData> TalentData
        {
            get { return talentData; }
        }

        #endregion

        #region Constructor Region

        public TalentDataManager()
        {
            talentData = new Dictionary<string, TalentData>();
        }

        #endregion

        #region Method Region

        public void FillTalents()
        {
            TalentData data = new TalentData
            {
                Name = "Bash",
                TalentPrerequisites = new string[0],
                LevelRequirement = 1,
                TalentType = TalentType.Activated,
                ActivationCost = 5,
                AllowedClasses = new string[] { "Fighter" },
                AttributeRequirements = new Dictionary<string, int>() { { "Strength", 10 } }
            };

            DamageEffect effect = DamageEffect.FromDamageEffectData(new DamageEffectData
            {
                TargetType = SpellClasses.TargetType.Enemy,
                AttackType = AttackType.Health,
                DamageType = DamageType.Crushing,
                DieType = DieType.D8,
                Modifier = 2,
                Name = "Bash",
                NumberOfDice = 2
            });

            data.Effects.Add(effect);

            talentData.Add("Bash", data);

            data = new TalentData()
            {
                Name = "Below The Belt",
                TalentPrerequisites = new string[0],
                LevelRequirement = 1,
                TalentType = TalentType.Activated,
                ActivationCost = 5,
                AllowedClasses = new string[] { "Rogue" },
                AttributeRequirements = new Dictionary<string, int>() { { "Dexterity", 10 } }
            };

            effect = DamageEffect.FromDamageEffectData(new DamageEffectData()
            {
                TargetType = SpellClasses.TargetType.Enemy,
                AttackType = AttackType.Health,
                DamageType = DamageType.Piercing,
                DieType = DieType.D4,
                Modifier = 2,
                Name = "Below The Belt",
                NumberOfDice = 3
            });

            data.Effects.Add(effect);

            talentData.Add("Below The Belt", data);
        }

        #endregion

        #region Virtual Method region
        #endregion
    }
}
```

We also need to make an adjustment to the FromDamageEffectData method of the DamageEffect class to use target types. Replace the DamageEffect class with the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using RpgLibrary.CharacterClasses;
namespace RpgLibrary.EffectClasses
{
    public class DamageEffect : BaseEffect
    {
        #region Field Region

        DamageType damageType;
        AttackType attackType;
        DieType dieType;
        int numberOfDice;
        int modifier;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        private DamageEffect()
        {
        }

        #endregion

        #region Method Region

        public static DamageEffect FromDamageEffectData(DamageEffectData data)
        {
            DamageEffect effect = new DamageEffect
            {
                name = data.Name,
                damageType = data.DamageType,
                attackType = data.AttackType,
                dieType = data.DieType,
                numberOfDice = data.NumberOfDice,
                modifier = data.Modifier,
                TargetType = data.TargetType
            };

            return effect;
        }

        #endregion

        #region Virtual Method Region

        public override void Apply(Entity entity)
        {
            int amount = modifier;
            for (int i = 0; i < numberOfDice; i++)
            {
                amount += Mechanics.RollDie(dieType);
            }

            foreach (Weakness weakness in entity.Weaknesses.Where(x => x.WeaknessType ==
damageType))
            {
                amount = weakness.Apply(amount);
            }

            foreach (Resistance resistance in entity.Resistances.Where(x => x.ResistanceType ==
damageType))
            {
                amount = resistance.Apply(amount);
            }

            if (amount < 1)
                amount = 1;

            switch (attackType)
            {
                case AttackType.Health:
                    entity.Health.Damage((ushort)amount);
                    break;
                case AttackType.Mana:
                    entity.Mana.Damage((ushort)amount);
                    break;
                case AttackType.Stamina:
                    entity.Stamina.Damage((ushort)amount);
                    break;
            }
        }

        #endregion
    }
}
```

If you build and run now you should be able to use spells and talents in combat. The damage will be applied correctly, and the action selection list will disappear when a talent or spell is selected, or the escape key has been released.

I initially planned on handling balancing armor and weapons in this tutorial. After further consideration, I will be putting that off to another tutorial. Instead, I am going to add a new feature to the combat engine. I'm going to add logging of what the player and mob do. It will queue the action, display for a few seconds, then move on to the next action. To start, add the following static field to the GameState class in the MGRpgLibrary.

```
protected readonly static Queue<string> _descriptions = new Queue<string>();
```

A queue is the perfect data structure for this action. You queue the items and then dequeue them after a fixed interval, drawing the head of the queue. I added it to GameState instead of BaseGameState because in some instances we will want to queue things from the library. I made it protected so all of the child states will have access to it. It is static because I will only ever want one of them.

The next step is to start logging what is going on. For now, it will be a simple, I hit you, you hit me affair. In the future I will update it so that it states how much damage you and the mob do. Add the following field and update the Update method of the CombatScreen to the following.

```
private double _queueTimer;

public override void Update(GameTime gameTime)
{
    _queueTimer += gameTime.ElapsedGameTime.TotalSeconds;

    if (_queueTimer > 2)
    {
        if (GameState._descriptions.Count > 0)
        {
            GameState._descriptions.Dequeue();
        }

        _queueTimer = 0;
    }

    if (!_displayActionTexture)
    {
        _scene.Update(gameTime, PlayerIndex.One);
    }

    if (GamePlayScreen.Player.Character.Entity.Health.CurrentValue <= 0)
    {
        StateManager.ChangeState(GameRef.GameOverScreen);
    }

    if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Space) && !
_displayActionTexture)
    {
        switch (_scene.SelectedIndex)
        {
            case 0:
                _queueTimer = 0;
                if (GamePlayScreen.Player.Character.Entity.Dexterity >= _mob.Entity.Dexterity)
                {
                    _descriptions.Enqueue($"You attack the {_mob.Entity.EntityClass}.");
                    _mob.Attack(GamePlayScreen.Player.Character.Entity);

                    if (_mob.Entity.Health.CurrentValue <= 0)
                    {
                        StateManager.PopState();
                        return;
                    }

                    _descriptions.Enqueue($"The {_mob.Entity.EntityClass} attacks you.");
                    _mob.DoAttack(GamePlayScreen.Player.Character.Entity);
                }
                else
                {
                    _descriptions.Enqueue($"The {_mob.Entity.EntityClass} attacks you.");
                    _mob.DoAttack(GamePlayScreen.Player.Character.Entity);
                    _descriptions.Enqueue($"You attack the {_mob.Entity.EntityClass}.");
                    _mob.Attack(GamePlayScreen.Player.Character.Entity);

                    if (_mob.Entity.Health.CurrentValue <= 0)
                    {
                        StateManager.PopState();
                        return;
                    }
                }
                break;
            case 1:
                _displayActionTexture = true;

                _actions.Clear();
                Entity entity = GamePlayScreen.Player.Character.Entity;

                if (entity.Mana.MaximumValue > 0)
                {
                    foreach (SpellData s in DataManager.SpellData.SpellData.Values)
                    {
                        foreach (string c in s.AllowedClasses)
                        {
                            if (c == entity.EntityClass && entity.Level >= s.LevelRequirement)
                            {
                                _actions.Add(s.Name);
                            }
                        }
                    }
                }
                else
                {
                    foreach (TalentData s in DataManager.TalentData.TalentData.Values)
                    {
                        foreach (string c in s.AllowedClasses)
                        {
                            if (c == entity.EntityClass && entity.Level >= s.LevelRequirement)
                            {
                                _actions.Add(s.Name);
                            }
                        }
                    }
                }
                return;
            case 2:
                Vector2 center = GamePlayScreen.Player.Sprite.Center;
                Vector2 mobCenter = _mob.Sprite.Center;
                _action = 0;

                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;
            }
        }

        if (_displayActionTexture)
        {
            if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Escape))
            {
                _displayActionTexture = false;
                return;
            }
            else if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Space))
            {
                DoAction();
            }
            else if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Down))
            {
                _action++;

                if (_action >= _actions.Count)
                {
                    _action = 0;
                }
            }
            else if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Up))
            {
                _action--;

                if (_action < 0)
                {
                    _action = _actions.Count - 1;
                }
            }
        }

        base.Update(gameTime);
    }
```

What the new code is time how much time has passed between frames of the game. If it is more than two seconds it checks to see if there is something in the queue. If there is, it removes the first item in the queue. Next it resets the timer to zero.

Later on when it is checking for what the player wants to do, it then resets the queue timer. Then, it queues a string saying that the player has attacked the mob. Then it queues the mob attacking the player. If the mob is faster than the player the order is reversed. For now, I don't do anything if the player has used a spell or talent. I will add that in a future tutorial.

The next step will be to render the queue. I did that in the Draw method of the CombatScreen, of course. Update that method as follows.

```
public override void Draw(GameTime gameTime)
{
    base.Draw(gameTime);
    if (_selected == null)
        _selected = Game.Content.Load<Texture2D>(@"GUI\rightarrowUp");

    GameRef.SpriteBatch.Begin();

    Rectangle menuDest = new Rectangle(0, 720 - _menuTexture.Height, _menuTexture.Width,
_menuTexture.Height);
    Rectangle hudDest = new Rectangle(0, menuDest.Top - _hudTexture.Height, _hudTexture.Width,
_hudTexture.Height);

    GameRef.SpriteBatch.Draw(_menuTexture, menuDest, Color.White);
    GameRef.SpriteBatch.Draw(_hudTexture, hudDest, Color.White);

    Rectangle dest = new Rectangle(10, hudDest.Top + 12, _healthBorderTexture.Width,
_healthBorderTexture.Height);
    GameRef.SpriteBatch.Draw(_healthBorderTexture, dest, Color.White);

    dest.X += 5;
    dest.Y += 2;
    dest.Height -= 5;
    dest.Width = (int)(dest.Width *
((float)GamePlayScreen.Player.Character.Entity.Health.CurrentValue /
GamePlayScreen.Player.Character.Entity.Health.MaximumValue)) - 10;

    GameRef.SpriteBatch.Draw(_healthTexture, dest, Color.Red);

    dest = new Rectangle(Game1.ScreenWidth - _healthBorderTexture.Width - 10, hudDest.Top + 12,
_healthBorderTexture.Width, _healthBorderTexture.Height);
    GameRef.SpriteBatch.Draw(_healthBorderTexture, dest, Color.White);

    dest.X += 5;
    dest.Y += 2;
    dest.Height -= 5;
    dest.Width = (int)(dest.Width * ((float)_mob.Entity.Health.CurrentValue /
_mob.Entity.Health.MaximumValue)) - 10;

    GameRef.SpriteBatch.Draw(_healthTexture, dest, Color.Red);

    _scene.Draw(gameTime, GameRef.SpriteBatch);

    if (_displayActionTexture)
    {
        GameRef.SpriteBatch.Draw(_actionTexture, new Rectangle(_actionTexture.Width, 0,
_actionTexture.Width, _actionTexture.Height), Color.White);
        Vector2 position = new Vector2(_actionTexture.Width + 50, 5);

        int count = 0;

        foreach (string s in _actions)
        {
            Color myColor = Color.Black;

            if (count == _action)
            {
                myColor = Color.White;
                GameRef.SpriteBatch.Draw(_selected, new Rectangle((int)position.X - 35,
(int)position.Y, _selected.Width, _selected.Height), Color.White);
            }

            GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"), s, position,
myColor);
            position.Y += FontManager.GetFont("testfont").LineSpacing + 5;

            count++;
        }
    }

    if (_descriptions.Count > 0)
    {
        string text = _descriptions.Peek();

        GameRef.SpriteBatch.DrawString(
            FontManager.GetFont("testfont"),
            text,
            new Vector2(10, Game1.ScreenHeight - FontManager.GetFont("testfont").LineSpacing),
            Color.White);
    }

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

What the new code does is check to see if there is an item in the queue. If there is it grabs its value into a local variable. It then draws the string at 10 pixels to the right of the edge of the screen and up the line spacing property pixels of the bottom of the screen.

The last thing I want to do in this tutorial is add an override of the Show method. When it is shown I want to reset the queue timer to 0. Add this override of the Show method.

```
protected override void Show()
{
    _queueTimer = 0;
    base.Show();
}
```

I am not going to bore you with details, but I added more equipment for the player to buy from the merchant. Mostly, it was calculating the source rectangles from the texture, and finding images I liked the look of for the items. Since you've seen similar code, I won't explain the details. Change the LoadWorld method to the following.

```
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 data)
        {
            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(
                    data.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)
        {
            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)
        {
            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)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetWeapon("Apprentice Staff"), "FullSheet", new
Rectangle(224, 1408, 32, 32)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetWeapon("Acolyte Staff"), "FullSheet", new
Rectangle(256, 1408, 32, 32)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetArmor("Leather Armor"), "FullSheet", new
Rectangle(1248, 1216, 32, 32)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetArmor("Chain Mail"), "FullSheet", new
Rectangle(1472, 1184, 32, 32)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetArmor("Studded Leather Armor"), "FullSheet",
new Rectangle(1984, 1120, 32, 32)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetArmor("Light Robes"), "FullSheet", new
Rectangle(992, 1216, 32, 32)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetArmor("Medium Robes"), "FullSheet", new
Rectangle(1024, 1216, 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)));
}
```

So, that is going to be it for this tutorial. I accomplished what I intended and 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