Shadow Monsters – MonoGame Tutorial Series
Chapter 12
Loading the Game

This tutorial series is about creating a Pokemon style game with the MonoGame Framework called Shadow Monsters. The tutorials will make more sense if you read them in order as each tutorial builds on the previous tutorials. You can find the list of tutorials on my blog: Shadow Monsters. The source code for each tutorial will be available as well. I will be using Visual Studio 2019 Community for the series. The code should compile on the 2013, 2015 and 2017 versions as well.

I want to mention though that the series is released as Creative Commons 3.0 Attribution. It means that you are free to use any of the code or graphics in your own game, even for commercial use, with attribution. Just give credit to Cynthia McMahon and add a link to my site, https://mygameprogrammingadventures.blogspot,com. Screenshots of your project and/or a video of game play would be appreciated.

I also want to mention that I assume you have a basic understanding of C# and MonoGame. If you don't I recommend that you learn basic C# and work with MonoGame a little. Enough to know the basics of fields, properties, methods, classes and the MonoGame framework.

In this tutorial we will reverse the process of the last tutorial and read a saved game from storage back in. Let's get started.

There are two approaches that can be taken for loading data back in. The first is that you can create an instance of a class and call a Load method on that instance. The second is that you can have a static method that returns the data. I will be using the later of the two.

We need to load the portal layer, collision layer, tile set, tile layers, shadow monsters, backpack, characters, character layer, tile map and player. I will add the load methods in that order. Before getting to the Load methods for loading I want to make a change to the Save method of the Player class. I just want to write out a delimiter before writing the backpack. Change the Save method of the Player class to the following.

```
internal void Save(BinaryWriter writer)
{
    StringBuilder b = new StringBuilder();

    b.Append(name);
    b.Append(",");
    b.Append(gender);
    b.Append(",");
    b.Append(mapName);
    b.Append(",");
    b.Append(tile.X);
    b.Append(",");
    b.Append(tile.Y);
    b.Append(",");
    b.Append(textureName);
    b.Append(",");
```

```
        b.Append(speed);
        b.Append(",");
        b.Append(sprite.CurrentAnimation);
        b.Append(",");
        b.Append(sprite.Position.X);
        b.Append(",");
        b.Append(sprite.Position.Y);

        writer.Write(b.ToString());
        writer.Write(shadowMonsters.Count);

        foreach (ShadowMonster a in shadowMonsters)
        {
            a.Save(writer);
            writer.Write(-1);
        }

        writer.Write(selected);

        foreach (ShadowMonster a in battleShadowMonsters)
        {
            if (a != null)
            {
                a.Save(writer);
                writer.Write(-1);
            }
            else
            {
                writer.Write("*");
                writer.Write(-1);
            }
        }

        writer.Write(-1);
        backpack.Save(writer);
    }
```

I also made a change to the LoadContent of the GamePlayState. When I added the tile set there was a mismatch between what was loaded and the name used. Change the LoadContent method of the GamePlayState.

```
    protected override void LoadContent()
    {
        MoveManager.FillMoves();
        ShadowMonsterManager.FromFile(@".\Content\ShadowMonsters.txt", content);
        Game1.Player.AddShadowMonster(ShadowMonsterManager.GetShadowMonster("water1"));
        Game1.Player.SetCurrentShadowMonster(0);
        Game1.Player.BattleShadowMonsters[0] = Game1.Player.GetShadowMonster(0);
        TileSet set = new TileSet();
        set.TextureNames.Add("tileset16-outdoors");
        set.Textures.Add(content.Load<Texture2D>(@"Tiles\tileset16-outdoors"));

        TileLayer groundLayer = new TileLayer(100, 100, 0, 1);
        TileLayer edgeLayer = new TileLayer(100, 100);
        TileLayer buildingLayer = new TileLayer(100, 100);
        TileLayer decorationLayer = new TileLayer(100, 100);

        for (int i = 0; i < 1000; i++)
        {
```

```
                decorationLayer.SetTile(random.Next(0, 100), random.Next(0, 100), 0,
random.Next(2, 4));
            }

            map = new TileMap(set, groundLayer, edgeLayer, buildingLayer, decorationLayer,
"level1");

            Character c = Character.FromString(GameRef,
"Paul,ninja_m,WalkDown,PaulHello,0,fire1,fire1,,,,,,dark1");
            c.Sprite.Position = new Vector2(2 * Engine.TileWidth, 2 * Engine.TileHeight);

            map.CharacterLayer.Characters.Add(new Point(2, 2), c);

            Merchant m = Merchant.FromString(GameRef,
"Bonnie,ninja_f,WalkLeft,BonnieHello,0,earth1,earth1,,,,,,");
            m.Sprite.Position = new Vector2(4 * Engine.TileWidth, 4 * Engine.TileHeight);
            m.Backpack.AddItem("Potion", 99);
            m.Backpack.AddItem("Antidote", 10);

            map.CharacterLayer.Characters.Add(new Point(4, 4), m);
            engine.SetMap(map);
            base.LoadContent();
        }
```

First, I added a Load method for portals instead of just reading the data in the portal layer. Add the following code to the Portal class.

```
        public static Portal Load(BinaryReader reader)
        {
            Portal p = new Portal
            {
                DestinationLevel = reader.ReadString(),
                SourceTile = new Point(reader.ReadInt32(), reader.ReadInt32()),
                DestinationTile = new Point(reader.ReadInt32(), reader.ReadInt32())
            };

            return p;
        }
```

The method requires a BinaryReader that will read in the data. I create a new instance of a Portal and then use the reader to initialize the values.

The PortalLayer will need to read the number of portals and then call the Load method of the Portal class for each portal to read in the portal. Add the following code to the PortalLayer class.

```
        public static PortalLayer Load(BinaryReader reader)
        {
            PortalLayer layer = new PortalLayer();

            int count = reader.ReadInt32();

            for (int i = 0; i < count; i++)
            {
                string r = reader.ReadString();
                Portal p = Portal.Load(reader);
                layer.portals.Add(r, p);
            }

            return layer;
```

```
}
```

The method requires a BinaryReader for reading in the data. It creates a new PortalLayer instance to read in the values. It reads in the number of portals on the map. It loops that many times, reads the key for the portal and calls the Load method of the Portal class. It then adds the Portal to the layer.

The next Load that I'm going to implement is the CollisionLayer. Add the following method to the CollisionLayer class.

```
internal static CollisionLayer Load(BinaryReader reader)
{
    CollisionLayer layer = new CollisionLayer();
    int count = reader.ReadInt32();

    for (int i = 0; i < count; i++)
    {
        CollisionType collision = (CollisionType)reader.ReadInt32();
        Point area = new Point(
            reader.ReadInt32(),
            reader.ReadInt32());

        layer.collisions.Add(area, collision);
    }

    return layer;
}
```

Like the previous methods this method requires a BinaryReader for reading in the values. It creates a new instance of CollisionLayer. Next it reads in the number of collisions. Now it loops that many times. It reads the collision type as an integer and casts it to a CollisionType type. It reads the tile that the collision is on. It adds the collision to the dictionary using the two values that were just read in. Finally it returns the layer.

Loading a TileSet is the next class that I will implement add the following code to the TileSet class.

```
public static TileSet Load(ContentManager content, BinaryReader reader)
{
    TileSet t = new TileSet();

    int count = reader.ReadInt32();

    for (int i = 0; i < count; i++)
    {
        t.imageName.Add(reader.ReadString());
        t.image.Add(content.Load<Texture2D>(
            @"Tiles\" + t.imageName[t.imageName.Count - 1]));
        reader.ReadInt32();
    }

    t.TilesWide = reader.ReadInt32();
    t.TilesHigh = reader.ReadInt32();
    t.TileWidth = reader.ReadInt32();
    t.TileHeight = reader.ReadInt32();
    t.sourceRectangles = new Rectangle[t.TilesWide * t.TilesHigh];
```

```
        int tile = 0;

        for (int y = 0; y < t.TilesHigh; y++)
        {
            for (int x = 0; x < t.TilesWide; x++)
            {
                t.sourceRectangles[tile] = new Rectangle(
                    x * t.TileWidth,
                    y * t.TileHeight,
                    t.TileWidth,
                    t.TileHeight);
                tile++;
            }
        }

        return t;
    }
```

This method requires a ContentManager to read in the tile set images and a BinaryReader. It creates a new TileSet. You need to know the number of images to be read in so I read that. Now I loop the number of times there are images. I add the image name to the list of image names. Next I use the ContentManager that was passed in to load the tile set image. I then read in TilesWide, TilesHigh, TileWidth and TileHeight. I initialize the source rectangle array. There is a local variable that will be the index of the source rectangle. I then create the source rectangles like before. Finally I return the tile set.

The next Load method that I'm going to implement is the TileLayer class. Add the following code to the TileLayer class.

```
    public static TileLayer Load(BinaryReader reader)
    {
        TileLayer data = new TileLayer(reader.ReadInt32(), reader.ReadInt32());

        for (int y = 0; y < data.Height; y++)
        {
            for (int x = 0; x < data.Width; x++)
            {
                data.SetTile(x, y, reader.ReadInt32(), reader.ReadInt32());
            }
        }

        return data;
    }
```

The method requires a BinaryReader parameter to read in the data. It creates a new TileLayer reading in the width and height of the layer. It then loops over the height and width of the map. It calls SetTile passing in the x coordinate, y coordinate along with the read in tile set index and tile index. Finally it returns the layer.

Since characters have, or can have, shadow monsters I implemented the load method for shadow monsters before characters. Add the following Load method to the ShadowMonster class.

```
        public static ShadowMonster Load(ContentManager content, string monster)
        {
            ShadowMonster s = new ShadowMonster();

            string[] parts = monster.Split(',');
            s.name = parts[0];
            s.displayName = parts[1];
            s.texture = content.Load<Texture2D>(@"ShadowMonsterImages\" + parts[1]);
            s.element = (ShadowMonsterElement)Enum.Parse(typeof(ShadowMonsterElement),
parts[2]);

            int.TryParse(parts[3], out int value);
            s.experience = value;

            int.TryParse(parts[4], out value);
            s.costToBuy = value;

            int.TryParse(parts[5], out value);
            s.level = value;

            int.TryParse(parts[6], out value);
            s.attack = value;

            int.TryParse(parts[7], out value);
            s.defense = value;

            int.TryParse(parts[8], out value);
            s.speed = value;

            int.TryParse(parts[9], out value);
            s.health = value;

            int.TryParse(parts[10], out value);
            s.currentHealth = value;

            bool.TryParse(parts[11], out bool value1);
            s.IsAsleep = value1;

            bool.TryParse(parts[12], out value1);
            s.IsConfused = value1;

            bool.TryParse(parts[13], out value1);
            s.IsParalyzed = value1;

            bool.TryParse(parts[14], out value1);
            s.IsPoisoned = value1;

            int.TryParse(parts[15], out int x);
            int.TryParse(parts[16], out int y);
            s.Source = new Rectangle(x, y, 64, 64);

            for (int i = 17; i < parts.Length; i++)
            {
                string[] moveParts = parts[i].Split(':');

                if (moveParts[0] != "None")
                {
                    IMove move = MoveManager.GetMove(moveParts[0]);
                    int.TryParse(moveParts[1], out value);
                    move.UnlockedAt = value;
```

```
            if (move.UnlockedAt <= s.Level)
            {
                move.Unlock();
            }

            s.knownMoves.Add(move.Name, move);
        }
    }

    return s;
}
```

The method takes a ContentManager and a string parameter. It is pretty much a copy/paste of the FromString method. It was improved a bit to use TryParse instead of Parse for integers.

Before getting to characters I need to implement loading a Backpack. Add this Load method to the Backpack class.

```
public static Backpack Load(BinaryReader reader)
{
    reader.ReadInt32();

    Backpack b = new Backpack();
    int count = reader.ReadInt32();

    for (int i = 0; i < count; i++)
    {
        string[] item = reader.ReadString().Split(':');
        reader.ReadInt32();
        b.AddItem(item[0], int.Parse(item[1]));
    }

    return b;
}
```

The method requires a BinaryReader to read the data. First it creates a backpack. It then grabs the number of items in the Backpack. Next it loops that many times. It reads the string that describes the item with the count and splits the string on a colon. Reads the delimiter. It then calls AddItem passing in the parts and returns the backpack.

The next step is to read in characters. I will start with the Character class. Add this Load method to the Character class.

```
public static Character Load(ContentManager content, BinaryReader reader)
{
    Character c = new Character();

    string data = reader.ReadString();
    string[] parts = data.Split(',');
    c.name = parts[0];
    c.textureName = parts[1];
    c.sprite = new AnimatedSprite(
        content.Load<Texture2D>(
            @"CharacterSprites\" + parts[1]),
```

```
                Game1.Animations);
            c.sprite.CurrentAnimation = (AnimationKey)Enum.Parse(typeof(AnimationKey),
parts[2]);
            c.conversation = parts[3];
            c.currentMonster = int.Parse(parts[4]);
            c.Battled = bool.Parse(parts[5]);

            reader.ReadInt32();

            for (int i = 0; i < 6; i++)
            {
                string avatar = reader.ReadString();

                if (avatar != "*")
                {
                    c.monsters[i] = ShadowMonster.Load(content, avatar);
                }

                reader.ReadInt32();
            }

            string giving = reader.ReadString();

            if (giving != "*")
            {
                c.givingMonster = ShadowMonster.Load(content, giving);
            }

            reader.ReadInt32();

            return c;
        }
```

This method takes a ContentManager parameter and a BinaryReader parameter. It creates a Character instance to return. It needs to read an integer to clear a previous delimiter. It then reads a string that describes the character. It is then split into the parts on the comma. The parts are then converted and assigned to the fields. Then I read the delimiter. It then loops six times to read the character's shadow monsters. It reads the string for the shadow monster. It it is not the asterisk it calls the Load method passing in the ContentManager and the string. It then reads the delimiter. It then does something similar for giving shadow monster and then returns the character.

The next Load to implement is the Merchant class. Add the following code to the Merchant class.

```
        new public static Merchant Load(ContentManager content, BinaryReader reader)
        {
            Merchant c = new Merchant
            {
                backpack = new Backpack()
            };

            string data = reader.ReadString();
            string[] parts = data.Split(',');
            reader.ReadInt32();

            c.name = parts[0];
            c.textureName = parts[1];
```

```
            c.sprite = new AnimatedSprite(
                content.Load<Texture2D>(
                    @"CharacterSprites\" + parts[1]),
                Game1.Animations);

            c.sprite.CurrentAnimation = (AnimationKey)Enum.Parse(typeof(AnimationKey),
parts[2]);
            c.conversation = parts[3];
            c.currentMonster = int.Parse(parts[4]);
            c.Battled = bool.Parse(parts[5]);

            for (int i = 0; i < 6; i++)
            {
                string avatar = reader.ReadString();

                if (avatar != "*")
                {
                    c.monsters[i] = ShadowMonster.Load(content, avatar);
                }

                reader.ReadInt32();
            }

            string giving = reader.ReadString();

            if (giving != "*")
            {
                c.givingMonster = ShadowMonster.Load(content, giving);
            }

            c.backpack = Backpack.Load(reader);
            return c;
        }
```

Because the method is static we can make it virtual and override it and call the base method. Instead we need to use the new keyword so that we don't get a warning from the compiler. The method requires a ContentManager and a BinaryReader. The method works the same as the Character class up until it needs to read the Backpack. It sets the Backpack using the Load method of the Backpack class. Finally it returns the merchant.

Now that we have the Load method for Characters and Merchants we can add the Load method for the CharacterLayer. Add the following code to the CharacterLayer class.

```
        public static CharacterLayer Load(ContentManager content, BinaryReader reader)
        {
            CharacterLayer layer = new CharacterLayer();

            int count = reader.ReadInt32();

            for (int i = 0; i < count; i++)
            {
                Character c = null;
                int charType = reader.ReadInt32();
                Point position = new Point(reader.ReadInt32(), reader.ReadInt32());

                if (charType == 1)
                {
                    c = Character.Load(content, reader);
```

```
            }
            else if (charType == 2)
            {
                c = Merchant.Load(content, reader);
            }

            layer.characters.Add(position, c);
        }
        return layer;
    }
```

The method requires a ContentManager parameter and a BinaryReader parameter. The method creates a CharacterLayer that will be returned. It reads the number of characters on the layer. It then loops that many times. There is a Character variable that will hold the character or merchant that will be read in. It then reads what type of character is being read in. Now it gets the tile that the character is on. If the character type is 1 it calls the Load method of the Character class and if it is 2 the Merchant class. The character is then added to the layer. Finally the layer is returned.

The next Load method that we will be implementing is the TileMap's Load method. Add the following code to the TileMap class.

```
    public static TileMap Load(ContentManager content, BinaryReader reader)
    {
        TileMap map = new TileMap();

        map.mapName = reader.ReadString();
        map.characterLayer = CharacterLayer.Load(content, reader);
        map.tileSet = TileSet.Load(content, reader);
        map.edgeLayer = TileLayer.Load(reader);
        map.groundLayer = TileLayer.Load(reader);
        map.decorationLayer = TileLayer.Load(reader);
        map.buildingLayer = TileLayer.Load(reader);
        map.portalLayer = PortalLayer.Load(reader);
        map.collisionLayer = CollisionLayer.Load(reader);

        map.mapWidth = map.groundLayer.Width;
        map.mapHeight = map.groundLayer.Height;

        return map;
    }
```

The method requires a ContentManager and BinaryReader parameter. It creates a new map using the parameterless constructor. It reads in the map name and assigns it to the mapName field. It then reads in the character layer, tile set, edge layer, ground layer, decoration layer, building layer, portal layer and collision layer. The same order that we saved them in the last tutorial. It then sets the mapWidth and mapHeight fields of the map. Finally it returns the map.

The last Load method to be implemented is the Player class. Add the following code to the Player class.

```
    public static Player Load(Game1 game, BinaryReader reader)
    {
        Player player = new Player(game);
        player.gameRef = game;
```

```
            string data = reader.ReadString();
            string[] parts = data.Split(',');

            player.name = parts[0];
            player.gender = bool.Parse(parts[1]);
            player.mapName = parts[2];
            player.tile = new Point(
                int.Parse(parts[3]),
                int.Parse(parts[4]));
            player.textureName = parts[5];
            player.speed = float.Parse(parts[6]);
            player.sprite = new AnimatedSprite(
                game.Content.Load<Texture2D>(parts[5]),
                Game1.Animations);
            player.sprite.CurrentAnimation = (AnimationKey)Enum.Parse(typeof(AnimationKey),
parts[7]);
            player.sprite.Position = new Vector2(float.Parse(parts[8]), float.Parse(parts[9]));

            int count = reader.ReadInt32();

            for (int i = 0; i < count; i++)
            {
                ShadowMonster a = ShadowMonster.Load(game.Content, reader.ReadString());
                reader.ReadInt32();

                player.shadowMonsters.Add(a);
            }

            player.selected = reader.ReadInt32();

            for (int i = 0; i < 6; i++)
            {
                string monster = reader.ReadString();
                reader.ReadInt32();

                if (monster != "*")
                {
                    ShadowMonster a = ShadowMonster.Load(game.Content, monster);
                    player.battleShadowMonsters[i] = a;
                }
            }

            player.backpack = Backpack.Load(reader);

            return player;
        }
```

The method requires a Game1 parameter and a BinaryReader. It creates a new Player object. It then assigns the gameRef field to the Game1 parameter passed in. It reads the string that describes the player and splits it into its parts. The parts are then assigned to their fields similar to the Character class. It then reads the number of shadow monsters that the play has. It loops that many times calling the Load method of the ShadowMonster class. It reads the delimiter and then adds the shadow monster to the player's collection of shadow monsters. Next it reads the select shadow monster. It loops six times reading the shadow monster and delimiter. If the value read in is not the asterisk it calls the Load method of the ShadowMonster class. It then calls the Load method of the Backpack to read in the player's items.

With all of the Load methods in place it is time to implement loading a game back in. That will be done if the player presses the F2 key. That will of course be done in the GamePlayState's Update method. Replace the existing Update method with the following code.

```
public override void Update(GameTime gameTime)
{
    engine.Update(gameTime);
    frameCount++;
    if (Xin.KeyboardState.IsKeyDown(Keys.W) && !inMotion)
    {
        motion.Y = -1;
        Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkUp;
        Game1.Player.Sprite.IsAnimating = true;
        inMotion = true;
        collision = new Rectangle(
            (int)Game1.Player.Sprite.Position.X,
            (int)Game1.Player.Sprite.Position.Y - Engine.TileHeight * 2,
            Engine.TileWidth,
            Engine.TileHeight);
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.S) && !inMotion)
    {
        motion.Y = 1;
        Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkDown;
        Game1.Player.Sprite.IsAnimating = true;
        inMotion = true;
        collision = new Rectangle(
            (int)Game1.Player.Sprite.Position.X,
            (int)Game1.Player.Sprite.Position.Y + Engine.TileHeight * 2,
            Engine.TileWidth,
            Engine.TileHeight);
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.A) && !inMotion)
    {
        motion.X = -1;
        Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
        Game1.Player.Sprite.IsAnimating = true;
        inMotion = true;
        collision = new Rectangle(
            (int)Game1.Player.Sprite.Position.X - Engine.TileWidth * 2,
            (int)Game1.Player.Sprite.Position.Y,
            Engine.TileWidth,
            Engine.TileHeight);
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.D) && !inMotion)
    {
        motion.X = 1;
        Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
        Game1.Player.Sprite.IsAnimating = true;
        inMotion = true;
        collision = new Rectangle(
            (int)Game1.Player.Sprite.Position.X + Engine.TileWidth * 2,
            (int)Game1.Player.Sprite.Position.Y,
            Engine.TileWidth,
            Engine.TileHeight);
    }

    if (motion != Vector2.Zero)
    {
```

```
                motion.Normalize();
                motion *= (Game1.Player.Sprite.Speed *
(float)gameTime.ElapsedGameTime.TotalSeconds);
                Rectangle pRect = new Rectangle(
                        (int)(Game1.Player.Sprite.Position.X + motion.X),
                        (int)(Game1.Player.Sprite.Position.Y + motion.Y),
                        Engine.TileWidth,
                        Engine.TileHeight);

                if (pRect.Intersects(collision))
                {
                    Game1.Player.Sprite.IsAnimating = false;
                    inMotion = false;
                    motion = Vector2.Zero;
                }

                foreach (Point p in engine.Map.CharacterLayer.Characters.Keys)
                {
                    Rectangle r = new Rectangle(
                        p.X * Engine.TileWidth,
                        p.Y * Engine.TileHeight,
                        Engine.TileWidth,
                        Engine.TileHeight);

                    if (r.Intersects(pRect))
                    {
                        motion = Vector2.Zero;
                        Game1.Player.Sprite.IsAnimating = false;
                        inMotion = false;
                    }
                }

                Vector2 newPosition = Game1.Player.Sprite.Position + motion;
                newPosition.X = (int)newPosition.X;
                newPosition.Y = (int)newPosition.Y;

                Game1.Player.Sprite.Position = newPosition;
                motion = Game1.Player.Sprite.LockToMap(
                    new Point(
                        map.WidthInPixels,
                        map.HeightInPixels),
                    motion);

                if (motion == Vector2.Zero)
                {
                    Vector2 origin = new Vector2(
                            Game1.Player.Sprite.Position.X + Game1.Player.Sprite.Origin.X,
                            Game1.Player.Sprite.Position.Y + Game1.Player.Sprite.Origin.Y);
                    Game1.Player.Sprite.Position = Engine.VectorFromOrigin(origin);
                    inMotion = false;
                    Game1.Player.Sprite.IsAnimating = false;
                }
            }

            if ((Xin.CheckKeyReleased(Keys.Space) ||
                Xin.CheckKeyReleased(Keys.Enter)) && frameCount >= 5)
            {
                frameCount = 0;
                foreach (Point s in engine.Map.CharacterLayer.Characters.Keys)
                {
```

```
            Character c = engine.Map.CharacterLayer.Characters[s];

            AnimationKey animation = Game1.Player.Sprite.CurrentAnimation;

            if (animation == AnimationKey.WalkLeft &&
                ((int)c.Sprite.Position.X > (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }

            if (animation == AnimationKey.WalkUp &&
                ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y > (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }

            if (animation == AnimationKey.WalkRight &&
                ((int)c.Sprite.Position.X < (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }

            if (animation == AnimationKey.WalkDown &&
                ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y < (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }

            float distance = Vector2.Distance(
                Game1.Player.Sprite.Origin + Game1.Player.Sprite.Position,
                c.Sprite.Origin + c.Sprite.Position);

            if (Math.Abs(distance) < Engine.TileWidth + Engine.TileWidth / 2)
            {
                manager.PushState(
                    (ConversationState)GameRef.ConversationState);

                GameRef.ConversationState.SetConversation(c);
                GameRef.ConversationState.StartConversation();
                break;
            }
        }
    }

    if (Xin.CheckKeyReleased(Keys.I))
    {
        manager.PushState(GameRef.ItemSelectionState);
    }

    if ((Xin.CheckKeyReleased(Keys.B)) && frameCount >= 5)
    {
        frameCount = 0;
        foreach (Point s in engine.Map.CharacterLayer.Characters.Keys)
        {
            Character c = engine.Map.CharacterLayer.Characters[s];
```

```
                    AnimationKey animation = Game1.Player.Sprite.CurrentAnimation;

                    if (animation == AnimationKey.WalkLeft &&
                        ((int)c.Sprite.Position.X > (int)Game1.Player.Sprite.Position.X ||
                            (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
                    {
                        continue;
                    }

                    if (animation == AnimationKey.WalkUp &&
                        ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                            (int)c.Sprite.Position.Y > (int)Game1.Player.Sprite.Position.Y))
                    {
                        continue;
                    }

                    if (animation == AnimationKey.WalkRight &&
                        ((int)c.Sprite.Position.X < (int)Game1.Player.Sprite.Position.X ||
                            (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
                    {
                        continue;
                    }

                    if (animation == AnimationKey.WalkDown &&
                        ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                            (int)c.Sprite.Position.Y < (int)Game1.Player.Sprite.Position.Y))
                    {
                        continue;
                    }

                    float distance = Vector2.Distance(
                        Game1.Player.Sprite.Origin + Game1.Player.Sprite.Position,
                        c.Sprite.Origin + c.Sprite.Position);

                    if (Math.Abs(distance) < Engine.TileWidth + Engine.TileWidth / 2 &&
c.Alive())
                    {
                        GameRef.StartBattleState.SetCombatants(Game1.Player, c);
                        manager.PushState(GameRef.StartBattleState);
                        break;
                    }
                }
            }

            if (Xin.CheckKeyReleased(Keys.F1))
            {
                using (Aes aes = Aes.Create())
                {
                    aes.IV = IV;
                    aes.Key = Key;

                    string path =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
                    path += "\\ShadowMonsters\\";

                    try
                    {
                        if (!Directory.Exists(path))
                        {
                            Directory.CreateDirectory(path);
```

```
            }
        }
        catch
        {
            // uh oh
        }

        try
        {
            ICryptoTransform encryptor = aes.CreateEncryptor(Key, IV);
            FileStream stream = new FileStream(
                path + "ShadowMonsters.sav",
                FileMode.Create,
                FileAccess.Write);
            using (CryptoStream cryptoStream = new CryptoStream(
                stream,
                encryptor,
                CryptoStreamMode.Write))
            {
                BinaryWriter writer = new BinaryWriter(cryptoStream);
                map.Save(writer);
                Game1.Player.Save(writer);
                writer.Close();
            }
            stream.Close();
            stream.Dispose();
        }
        catch
        {
            // uh oh
        }
    }
}
if (Xin.CheckKeyReleased(Keys.F2))
{
    using (Aes aes = Aes.Create())
    {
        aes.IV = IV;
        aes.Key = Key;

        string path = Environment.GetFolderPath(
            Environment.SpecialFolder.ApplicationData);
        path += "\\ShadowMonsters\\";

        try
        {
            ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
            FileStream stream = new FileStream(
                path + "ShadowMonsters.sav",
                FileMode.Open,
                FileAccess.Read);
            using (CryptoStream cryptoStream = new CryptoStream(
                stream,
                decryptor,
                CryptoStreamMode.Read))
            {
                BinaryReader reader = new BinaryReader(cryptoStream);
                map  = TileMap.Load(content, reader);
                Game1.Player = Player.Load(GameRef, reader);
                reader.Close();
```

```
                }
                stream.Close();
                stream.Dispose();
            }
            catch (Exception exc)
            {
                exc.ToString();
            }
        }
    }
```

The loading code is very similar to the saving code. It creates a new Aes instance. It assigns the initialization vector and key to the byte arrays that were used for saving. It creates a decryptor using the initialization vector and key as well. It gets the path to the save game location. There is then a try-catch block around the loading code. It creates a FileStream for opening and read access. It then creates a CryptoStream using the file stream and decryptor for read access. A BinaryReader is created using the CryptoStream. The map field is set to the loaded TileMap and player is set to the loaded player. The reader is then closed. The file stream is then closed and disposed. The error is then caught in the catch block. We still need to handle the error effectively in a future tutorial.

I'm going to wrap this tutorial up here. I will be starting work on the next tutorial shortly. Keep checking back on the blog for news on that tutorial. I hope to have it up in the next week or so.

I wish you the best in your MonoGame Programming Adventures!