A Summoner’s Tale
Chapter E
Who’s That Girl?

Welcome to my Summoner’s Tale tutorial series on creating a Pokémon-inspired game with MonoGame. I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. The code should work on previous versions of MonoGame and Visual Studio. I plan on creating the editors on macOS and Windows. I’m unfamiliar with Linux, so a few projects may not be done for that platform.

The tutorials will make more sense if they are read in order. You can find the list of tutorials on my web blog, A Summoner’s Tale page. In addition to the PDFs, I will make the code for each tutorial available on GitHub here: https://github.com/Synammon/summoners-tale. It will also be included on the page that links to the tutorials.

This tutorial will add a non-player character to the game, a girl. Her name is Rio, and she dances on the sand. We have a bit of the chicken and the egg situation going on. The shared project needs to know about characters. The Psilibrary needs to know about characters as well. The SummonersTale gets full access to the classes in the Psilibrary. The PSilibrary, on the other hand, has no access to the SummonersTale project. So, how are we ever going to resolve this? With one very simple, yet very powerful, word: Interfaces. In the Psilibrary project, we are going to create an interface that holds the absolute bare minimum it requires. Okay, maybe a little bit more than the bare minimum. In the SummonersTale project, we implement the interface in any needed classes. The interface allows the sharing of member data. Well, how does that exactly work, Cynthia? Interfaces are only contracts. Yes, they are contracts, but they can act like fields. It will be so much clearer when you see it in action.

With the main theory lesson done, let’s dig into the tutorial. As I said, the Psilibrary needs an interface to access the details it needs to know about characters. So, I added an interface called ICharacter. First, I created a folder. Then, right-click the PSilibrary project, select Add and then New Folder, and name this new folder Characters. Next, right-click the Characters folder, select Add and then New Item. Finally, navigate the list to Interface, and call the interface ICharacter. Here is the code for that interface.

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

namespace Psilibrary.Characters
{
    public interface ICharacter
    {
        string Name { get; }
        bool Enabled { get; set; }
        bool Visible { get; set; }
        Vector2 Position { get; set; }
        Point Tile { get; set; }
        void Update(GameTime gameTime);
        void Draw(SpriteBatch spriteBatch);
    }
}
```

Okay, not that scar, huh? It is just a collection of properties and methods that must be implemented by any class that implements it. There is a get-only property for the name of the character. That doesn't mean that the class implementing it has to do that. It is just the minimum requirement. There are Enabled and Visible properties. A position in pixels and a position in tiles. Then comes the two methods, Update and Draw.

Nothing big or scary here as well. In Eyes of the Dragon, I implemented an ILayer interface for the map layers. This project does not use the ILayer interface. I was torn, though, about switching this project to use ILayer. I've decided that it is worth the hassle to do it. First, we need to add the ILayer interface. Right click the TileEngine folder in Psilibrary, select Add and then New Item. Scroll down the list until you get to interface. Name this new interface ILayer. Here is the code.

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

namespace Psilibrary.TileEngine
{
    public interface ILayer
    {
        void Update(GameTime gameTime);
        void Draw(SpriteBatch spriteBatch, Camera camera, List<TileSet> tilesets);
    }
}
```

So it has an Update and Draw method that must be implemented by any class that implements it. Now, we will add a layer that implements this interface. Right-click the TileEngine folder in the Psilibrary project, select Add and then Class. Name this new class CharacterLayer. Here is the code for that class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Psilibrary.Characters;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Psilibrary.TileEngine
{
    public class CharacterLayer : ILayer
    {
        public List<ICharacter> Characters { get; private set; } = new();

        public CharacterLayer() { }

        public void Update(GameTime gameTime)
        {
            foreach (var character in Characters.Where(x => x.Enabled))
            {
                character.Update(gameTime);
            }
        }

        public void Draw(SpriteBatch spriteBatch, Camera camera, List<TileSet>
tilesets)
        {
            foreach (var character in Characters.Where(x => x.Visible))
            {
                character.Draw(spriteBatch);
            }
        }
    }
}
```

There is a List<ICharacter> that will hold our characters. There is an explicit constructor. The Update method loops over the Enabled property in a LINQ Where clause. It does the same in the Draw method using the Visible property. So, as long as our characters implement the interface, they have Draw and Update methods that we can call, and they can be rendered on the map. Pretty neat, huh?

Wow. We are just speeding through this tutorial. I need to find something big and scary. I know! We can add the new layer to the map. That is going to be a little scary because we need to rip the guts out of the TileEngine class and adjust the TileLayer class. Let's start with the TileLayer class, since it is the easier of the two. Replace it with the following code.

```
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework;
using System.IO;
using System;
using System.Collections.Generic;

namespace Psilibrary.TileEngine
{
    public struct Tile
    {
        #region Property Region
        public int TileSet { get; set; }
        public int TileIndex { get; set; }

        #endregion

        #region Constructor Region

        public Tile()
        {
            TileSet = -1;
            TileIndex = -1;
        }

        public Tile(int set, int index)
        {
            TileSet = set;
            TileIndex = index;
        }

        #endregion
    }

    public class TileLayer : ILayer
    {
        #region Field Region

        [ContentSerializer]
        readonly Tile[] tiles;

        int width;
        int height;

        Point cameraPoint;
        Point viewPoint;
        Point min;
        Point max;
        Rectangle destination;

        #endregion

        #region Property Region

        public bool Enabled { get; set; }

        public bool Visible { get; set; }

        [ContentSerializer]
        public int Width
        {
            get { return width; }
            private set { width = value; }
        }

        [ContentSerializer]
        public int Height
        {
            get { return height; }
            private set { height = value; }
        }

        #endregion

        #region Constructor Region

        private TileLayer()
        {
            Enabled = true;
            Visible = true;
        }

        public TileLayer(Tile[] tiles, int width, int height)
            : this()
        {
            this.tiles = (Tile[])tiles.Clone();
            this.width = width;
            this.height = height;
        }

        public TileLayer(int width, int height)
            : this()
        {
            tiles = new Tile[height * width];
            this.width = width;
            this.height = height;

            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    tiles[y * width + x] = new Tile();
                }
            }
        }

        public TileLayer(int width, int height, int set, int index)
            : this()
        {
            tiles = new Tile[height * width];
            this.width = width;
            this.height = height;

            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    tiles[y * width + x] = new Tile(set, index);
                }
            }
        }

        #endregion

        #region Method Region

        public Tile GetTile(int x, int y)
        {
            if (x < 0 || y < 0)
                return new Tile();

            if (x >= width || y >= height)
                return new Tile();

            return tiles[y * width + x];
        }

        public void SetTile(int x, int y, int tileSet, int tileIndex)
        {
            if (x < 0 || y < 0)
                return;

            if (x >= width || y >= height)
                return;

            tiles[y * width + x] = new Tile(tileSet, tileIndex);
        }

        public void Update(GameTime gameTime)
        {
            if (!Enabled)
                return;
        }

        public void Draw(SpriteBatch spriteBatch, Camera camera, List<TileSet>
tileSets)
        {
            if (!Visible)
                return;

            cameraPoint = Engine.VectorToCell(camera.Position);
            viewPoint = Engine.VectorToCell(
                new Vector2(
                    (camera.Position.X + Engine.ViewportRectangle.Width),
                    (camera.Position.Y + Engine.ViewportRectangle.Height)));

            min.X = Math.Max(0, cameraPoint.X - 1);
            min.Y = Math.Max(0, cameraPoint.Y - 1);
            max.X = Math.Min(viewPoint.X + 1, Width);
            max.Y = Math.Min(viewPoint.Y + 1, Height);

            destination = new Rectangle(0, 0, Engine.TileWidth, Engine.TileHeight);
            Tile tile;

            for (int y = min.Y; y < max.Y; y++)
            {
                destination.Y = y * Engine.TileHeight;

                for (int x = min.X; x < max.X; x++)
                {
                    tile = GetTile(x, y);

                    if (tile.TileSet == -1 || tile.TileIndex == -1)
                        continue;
                    destination.X = x * Engine.TileWidth;

                    if (tileSets[0].TileSheets.Count > 0)
                    {
                        spriteBatch.Draw(
                            tileSets[0].TileSheets[tile.TileSet].Texture,
                            destination,
tileSets[0].TileSheets[tile.TileSet].SourceRectangles[tile.TileIndex],
                            Color.White);
                    }
                }
            }
        }

        #endregion
    }
}
```

The changes were that it now implements the ILayer method. Because it does, the signature of the Draw method changed. It no longer receives a GameTime parameter and it receives a list of tilesets, required by the interface. We handle tile sets differently, they are handled by the tile sheets. So, when comes to that we just use 0 for the tile set index.

Okay, TileMap. It is going to be a little scarier than the tile layer. The big issue is we have four layers with get and set accessors. We need to rip those out and add new ones. We also need to add a list of layers. So, replace the TileMap class with the following.

```
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Psilibrary.TileEngine
{
    public class TileMap
    {
        #region Field Region

        string mapName;
        List<ILayer> layers = new();

        int mapWidth;
        int mapHeight;

        List<TileSet> tileSets = new();

        #endregion

        #region Property Region
        [ContentSerializer(Optional = true)]
        public string MapName
        {
            get { return mapName; }
            private set { mapName = value; }
        }

        [ContentSerializer]
        public List<TileSet> TileSets
        {
            get { return tileSets; }
            set { tileSets = value; }
        }

        [ContentSerializer]
        public List<ILayer> Layers
        {
            get { return layers; }
            set { layers = value; }
        }

        [ContentSerializer]
        public int MapWidth
        {
            get { return mapWidth; }
            private set { mapWidth = value; }
        }

        [ContentSerializer]
        public int MapHeight
        {
            get { return mapHeight; }
            private set { mapHeight = value; }
        }

        public int WidthInPixels
        {
            get { return mapWidth * Engine.TileWidth; }
        }

        public int HeightInPixels
        {
            get { return mapHeight * Engine.TileHeight; }
        }

        #endregion

        #region Constructor Region

        private TileMap()
        {
        }

        private TileMap(List<TileSet> tileSets, string mapName)
            : this()
        {
            this.tileSets = tileSets;
            this.mapName = mapName;
        }

        public TileMap(
            List<TileSet> tileSets,
            List<ILayer> layers,
            string mapName)
            : this(tileSets, mapName)
        {
            this.layers = layers;
            this.tileSets = tileSets;

            TileLayer layer = (TileLayer)layers.Where(x => x is
TileLayer).FirstOrDefault();

            mapWidth = layer.Width;
            mapHeight = layer.Height;
        }

        #endregion

        #region Method Region

        public void Update(GameTime gameTime)
        {
            foreach (ILayer layer in this.layers)
            {
                layer.Update(gameTime);
            }
        }

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera,
bool debug = false)
        {
            if (WidthInPixels >= Engine.TargetWidth || debug)
            {
                spriteBatch.Begin(
                    SpriteSortMode.Deferred,
                    BlendState.AlphaBlend,
                    SamplerState.PointClamp,
                    null,
                    null,
                    null,
                    camera.Transformation);
            }
            else
            {
                Matrix m = Matrix.CreateTranslation(
                    new Vector3((Engine.TargetWidth) / 2, (Engine.TargetHeight -
HeightInPixels) / 2, 0));
                spriteBatch.Begin(
                    SpriteSortMode.Deferred,
                    BlendState.AlphaBlend,
                    SamplerState.PointClamp,
                    null,
                    null,
                    null,
                    m);
            }

            foreach (ILayer layer in this.layers)
            {
                layer.Draw(spriteBatch, camera, TileSets);
            }

            spriteBatch.End();
        }

        #endregion
    }
}
```

It's very similar to Eyes of the Dragon. I really love both methods. I'm just thinking that since will parallel a little, might as well make them closer together. The change the breaks things the most is that we no longer have separate fields for the layers and the tile sets. They are now List<T> fields. The properties were similarly adjusted, removing what we don't need and adding what we do.

The constructors changed from their previous versions. The second now takes a List<TileSet> as its first parameter. It then sets the fields based on the parameters. The third changed the most out of the two. It now takes a List<TileSet> and a List<ILayer> for the layers. It sets the fields based on the parameters. To determine the height and width of the map I need to find the first TileLayer. So, it is imperative that if you use this constructor that you include a TileLayer. If one isn't found, it will die a horrible, fiery death. So, avoid doing it!

The Update method now loops over the layers and calls their Update method. The Draw method does the same but calling Draw method, of course. An ounce of pain now, for a bucket of relief in the future.

It is time to implement the Character class. That will be do in the SummonersTale project. Right click the SummonersTale project, select Add and then Class. Name this new class Character. Here is the code for that class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Psilibrary.Characters;
using SummonersTale.SpriteClasses;
using SummonersTale.Sprites;
using System;
using System.Collections.Generic;
using System.Text;

namespace SummonersTale
{
    public class Character : ICharacter
    {
        private string _name;
        private AnimatedSprite _sprite;
        private string _spriteName;
        public string Name => _name;

        public bool Enabled { get; set; }
        public bool Visible { get; set; }
        public Vector2 Position { get; set; }
        public Point Tile { get; set; }

        private Character()
        {
            Enabled = true;
            Visible = true;
            Position = new();
            Tile = new();
        }

        public Character(string name, AnimatedSprite animatedSprite, string
spriteName)
        {
            _name = name;
            _sprite = animatedSprite;
            _spriteName = spriteName;
        }

        public void Draw(SpriteBatch spriteBatch)
        {
            _sprite.Draw(spriteBatch);
        }

        public void Update(GameTime gameTime)
        {
            _sprite.Position = Position;
            _sprite.Update(gameTime);
        }
    }
}
```

I added a few fields to the class. There is one for the name, an animated sprite, and a sprite name. There are then the properties that we talked about earlier. I added a private constructor that takes no parameters. It initializes the properties. There is a second constructor that takes a name, sprite, and sprite name. It assigns the properties to the values passed in. The Draw method calls the Draw method of the sprite. I needed to assign the Position property of the sprite as the Position property of the character to get it where I wanted it.

I did have to make an adjustment to the TileMap class. Currently, we only draw tile layers. We need to adjust it to draw CharacterLayers as well. It is just a simple matter of adding an else clause to the if. Replace the Draw method of the TileMap class with the following version, and add the AddLayer method.

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera, bool
debug = false)
{
    if (WidthInPixels >= Engine.TargetWidth || debug)
    {
        spriteBatch.Begin(
            SpriteSortMode.Deferred,
            BlendState.AlphaBlend,
            SamplerState.PointClamp,
            null,
            null,
            null,
            camera.Transformation);
    }
    else
    {
        Matrix m = Matrix.CreateTranslation(
            new Vector3((Engine.TargetWidth) / 2, (Engine.TargetHeight -
HeightInPixels) / 2, 0));
        spriteBatch.Begin(
            SpriteSortMode.Deferred,
            BlendState.AlphaBlend,
            SamplerState.PointClamp,
            null,
            null,
            null,
            m);
    }

    foreach (ILayer layer in this.layers)
    {
        if (layer is TileLayer || layer is CharacterLayer)
            layer.Draw(spriteBatch, camera, TileSets);
    }

    spriteBatch.End();
}

public void AddLayer(ILayer layer)
{
    this.layers.Add(layer);
}
```

Nothing big and scary there either. The last change to get the characters on the screen is to create a character layer and add it to the map. You do that in the LoadContent method of the GamePlayState. Replace that method with the following version.

```
protected override void LoadContent()
{
    base.LoadContent();

    Texture2D texture = Game.Content.Load<Texture2D>(@"Tiles/tileset1");

    TileSheet sheet = new(texture, "base", new(20, 18, 32, 32));
    List<TileSet> tilesets = new();
    tilesets.Add(new(sheet));

    TileLayer layer = new(100, 100);
    List<ILayer> layers= new();
    layers.Add(layer);

    _tileMap = new(tilesets, layers, "test");
    Dictionary<string, Animation> animations = new();

    Animation animation = new(3, 32, 32, 0, 0) { CurrentFrame = 0, FramesPerSecond =
8 };
    animations.Add("walkdown", animation);

    animation = new(3, 32, 32, 0, 32) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkleft", animation);

    animation = new(3, 32, 32, 0, 64) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkright", animation);

    animation = new(3, 32, 32, 0, 96) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkup", animation);

    texture = Game.Content.Load<Texture2D>(@"PlayerSprites/femalepriest");

    sprite = new(texture, animations)
    {
        CurrentAnimation = "walkdown",
        IsActive = true,
        IsAnimating = true,
    };

    texture = Game.Content.Load<Texture2D>(@"PlayerSprites/femalefighter");

    AnimatedSprite rio = new(texture, animations)
    {
        CurrentAnimation = "walkdown",
        IsAnimating = true,
    };

    CharacterLayer chars = new();

    chars.Characters.Add(
        new Character("Rio", rio, "femalefighter")
        {
            Position = new(320, 320),
            Tile = new(10, 10),
            Visible = true,
            Enabled = true,
        });

    _tileMap.AddLayer(chars);

    rightButton = new(Game.Content.Load<Texture2D>("GUI/g21245"), ButtonRole.Menu)
    {
        Position = new(80, Settings.BaseHeight - 80),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    rightButton.Down += RightButton_Down;
    Controls.Add(rightButton);

    upButton = new(Game.Content.Load<Texture2D>("GUI/g21263"), ButtonRole.Menu)
    {
        Position = new(48, Settings.BaseHeight - 48 - 64),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    upButton.Down += UpButton_Down;
    Controls.Add(upButton);

    downButton = new(Game.Content.Load<Texture2D>("GUI/g21272"), ButtonRole.Menu)
    {
        Position = new(48, Settings.BaseHeight - 48),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    downButton.Down += DownButton_Down;
    Controls.Add(downButton);

    leftButton = new(Game.Content.Load<Texture2D>("GUI/g22987"), ButtonRole.Menu)
    {
        Position = new(16, Settings.BaseHeight - 80),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    leftButton.Down += LeftButton_Down;

    Controls.Add(leftButton);
}
```

What has changed here? Well, after creating the player’s sprite, I load the female fighter texture. Then, I created an animated sprite for the character walking down and is animating to show that the layer is updating. Next, I create a new character. I pass in the name of the character, the sprite, and the name of the texture. I position it in tile (10, 10) and at position (320, 320). It is also Visible and Enabled. I then add the layer to the map.

```
protected override void LoadContent()
{
    base.LoadContent();

    Texture2D texture = Game.Content.Load<Texture2D>(@"Tiles/tileset1");

    TileSheet sheet = new(texture, "base", new(20, 18, 32, 32));
    List<TileSet> tilesets = new();
    tilesets.Add(new(sheet));

    TileLayer layer = new(100, 100, 0, 0);
    List<ILayer> layers= new();
    layers.Add(layer);

    _tileMap = new(tilesets, layers, "test");
    Dictionary<string, Animation> animations = new();

    Animation animation = new(3, 32, 32, 0, 0) { CurrentFrame = 0, FramesPerSecond =
8 };
    animations.Add("walkdown", animation);

    animation = new(3, 32, 32, 0, 32) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkleft", animation);

    animation = new(3, 32, 32, 0, 64) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkright", animation);

    animation = new(3, 32, 32, 0, 96) { CurrentFrame = 0, FramesPerSecond = 8 };
    animations.Add("walkup", animation);

    texture = Game.Content.Load<Texture2D>(@"PlayerSprites/femalepriest");

    sprite = new(texture, animations)
    {
        CurrentAnimation = "walkdown",
        IsActive = true,
        IsAnimating = true,
    };

    texture = Game.Content.Load<Texture2D>(@"PlayerSprites/femalefighter");

    AnimatedSprite rio = new(texture, animations)
    {
        CurrentAnimation = "walkdown",
        IsAnimating = true,
    };

    CharacterLayer chars = new();

    chars.Characters.Add(
        new Character("Rio", rio, "femalefighter")
        {
            Position = new(320, 320),
            Tile = new(10, 10),
            Visible = true,
            Enabled = true,
        });

    _tileMap.AddLayer(chars);

    rightButton = new(Game.Content.Load<Texture2D>("GUI/g21245"), ButtonRole.Menu)
    {
        Position = new(80, Settings.BaseHeight - 80),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    rightButton.Down += RightButton_Down;
    Controls.Add(rightButton);

    upButton = new(Game.Content.Load<Texture2D>("GUI/g21263"), ButtonRole.Menu)
    {
        Position = new(48, Settings.BaseHeight - 48 - 64),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    upButton.Down += UpButton_Down;
    Controls.Add(upButton);

    downButton = new(Game.Content.Load<Texture2D>("GUI/g21272"), ButtonRole.Menu)
    {
        Position = new(48, Settings.BaseHeight - 48),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    downButton.Down += DownButton_Down;
    Controls.Add(downButton);

    leftButton = new(Game.Content.Load<Texture2D>("GUI/g22987"), ButtonRole.Menu)
    {
        Position = new(16, Settings.BaseHeight - 80),
        Size = new(32, 32),
        Text = "",
        Color = Color.White,
    };

    leftButton.Down += LeftButton_Down;

    Controls.Add(leftButton);
}
```

If you build and run now, when you get to two big, ugly, hairy, smelly errors. They are in the editor. One is the editor itself, and the other is game. The problem in the editor itself is that we are creating a map in the LoadContent method using the old parameter. Thankfully, we can rip that code out entirely. Replace the LoadContent method of the Editor class with the following.

```
protected override void LoadContent()
{
    _spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here

    Services.AddService(typeof(SpriteBatch), _spriteBatch);

    _mainForm = new(this, Vector2.Zero, new(1920, 1080))
    {
        FullScreen = true,
        Title = "A Summoner's Tale Editor"
    };

    _menuForm = new(this, new(1920 + 80, 0), new(80, 1080))
    {
        FullScreen = true,
        Title = ""
    };

    manager.ChangeState(_mainForm);
}
```

There is a similar problem in the editor. Replace the LoadContent method of MainForm in the SummonersTale project.

```
protected override void LoadContent()
{
    base.LoadContent();

    GraphicsDeviceManager gdm = Game.Services.GetService<GraphicsDeviceManager>();

    _menuButton = new(content.Load<Texture2D>(@"GUI/g21688"), ButtonRole.Menu)
    {
        Text = "",
        Position = new(gdm.PreferredBackBufferWidth - 84, 20)
    };

    _menuButton.Click += MenuButton_Click;

    _renderTarget2D = new(GraphicsDevice, 1280, 1080);

    _mapDisplay = new(null, 40, 32)
    {
        HasFocus = false,
        Position = new(0, 0)
    };

    TileSheet sheet = new(content.Load<Texture2D>(@"Tiles/Overworld"), "test",
new(40, 36, 16, 16));
    TileSet set = new(sheet);

    List<TileSet> tileSets = new()
    {
        set
    };

    TileLayer ground = new(100, 100, 0, 0);

    List<ILayer> layers = new()
    {
        ground
    };

    TileMap tileMap = new(tileSets, layers, "test");

    _mapDisplay.SetMap(tileMap);

    Color[] data = new Color[1];
    data[0] = Color.Transparent;

    Texture2D b = new(GraphicsDevice, 1, 1);
    b.SetData(data);

    _fileForm = new(Game, Vector2.Zero, Size);
}
```

If you build and run now, you will see a second sprite on the map that is animating. Pretty cool, huh? Well, there is one problem. You can walk on top of her. Not cool! Let's fix that. Replace the Update method with the following version.

```
public override void Update(GameTime gameTime)
{
    Controls.Update(gameTime);

    sprite.Update(gameTime);
    _tileMap.Update(gameTime);

    if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.A) && !inMotion)
    {
        MoveLeft();
    }
    else if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.D) && !inMotion)
    {
        MoveRight();
    }

    if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.W) && !inMotion)
    {
        MoveUp();
    }
    else if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.S) && !inMotion)
    {
        MoveDown();
    }

    if (motion != Vector2.Zero)
    {
        motion.Normalize();
    }
    else
    {
        inMotion = false;
        return;
    }

    if (!sprite.LockToMap(new(99 * Engine.TileWidth, 99 * Engine.TileHeight), ref
motion))
    {
        inMotion = false;
        return;
    }

    Vector2 newPosition = sprite.Position + motion * speed *
(float)gameTime.ElapsedGameTime.TotalSeconds;

    Rectangle nextPotition = new Rectangle(
        (int)newPosition.X,
        (int)newPosition.Y,
        Engine.TileWidth,
        Engine.TileHeight);
    if (nextPotition.Intersects(collision))
    {
        inMotion = false;
        motion = Vector2.Zero;
        sprite.Position = new((int)sprite.Position.X, (int)sprite.Position.Y);
        return;
    }

    if (_tileMap.PlayerCollides(nextPotition))
    {
        inMotion = false;
        motion = Vector2.Zero;
        return;
    }

    sprite.Position = newPosition;
    sprite.Tile = Engine.VectorToCell(newPosition);

    _camera.LockToSprite(sprite, _tileMap);

    base.Update(gameTime);
}
```

What has changed here? Well, in an if statement, I call a new method PlayerCollides on the tile map that will check to see if there is a collision between the player and the characters. If there is, I set inMotion to false, the motion vector to the zero vector and exit the method. That leads us to the PlayerCollides method. Add this method to the TileMap class.

```
public bool PlayerCollides(Rectangle nextPotition)
{
    CharacterLayer layer = layers.Where(x => x is CharacterLayer).FirstOrDefault()
as CharacterLayer;

    if (layer != null)
    {
        foreach (var character in layer.Characters)
        {
            Rectangle rectangle = new(
                new(
                    character.Tile.X * Engine.TileWidth,
                    character.Tile.Y * Engine.TileHeight),
                new(
                    Engine.TileWidth,
                    Engine.TileHeight));
            if (rectangle.Intersects(nextPotition))
            {
                return true;
            }
        }
    }

    return false;
}
```

The first step is to get the character layer. I do that using a Where clause on the map layers, using the FirstOrDefault clause and casting it to a CharacterLayer. If that is not null, I cycle over all of the characters on the map. I then create a rectangle that represents the tile the character is on. If that intersects with the tile the player is trying to enter, I return true. If no collision is found, I return false.

That is it for getting a character on the map and collision detection between the player and the characters. So, I'm most definitely not going to dive any further into this tutorial. I think I've fed you more than enough for one day. I covered a lot, and I don't want to overload you. I encourage you to keep visiting my blog for the latest news on my tutorials.

Good luck with your game programming adventures.
Cynthia