A Summoner's Tale – MonoGame Tutorial Series

Chapter 14

Changing Maps

This tutorial series is about creating a Pokemon style game with the MonoGame Framework called A Summoner's Tale. 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 web site: A Summoner's Tale. The source code for each tutorial will be available as well. I will be using Visual Studio 2013 Premium for the series. The code should compile on the 2013 Express version and Visual Studio 2015 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 add a link to my site, http://gameprogrammingadventures.org, and credit to Cynthia McMahon.

In this tutorial I'm going to add the ability for the player to switch between maps. I will be adding a small building that the player can enter. To do this in my game I added in a new layer that I called a portal layer. I defined a portal as a tile that leads somewhere else. It does not have to lead to a different map but most often they will. So, you can also use a portal to move the player from one position on the map to a different position on the map, like a teleporter.

Let's get started. Open up your solution from the last time. Right click the TileEngine folder, select Add and then Class. Name this new class Portal. This class defines the properties and methods of a portal. Here is the code for this class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;

namespace Avatars.TileEngine
{
    public class Portal
    {
        #region Field Region

        Point sourceTile;
        Point destinationTile;
        string destinationLevel;

        #endregion

        #region Property Region
```

```
        [ContentSerializer]
        public Point SourceTile
        {
            get { return sourceTile; }
            private set { sourceTile = value; }
        }

        [ContentSerializer]
        public Point DestinationTile
        {
            get { return destinationTile; }
            private set { destinationTile = value; }
        }

        [ContentSerializer]
        public string DestinationLevel
        {
            get { return destinationLevel; }
            private set { destinationLevel = value; }
        }

        #endregion

        #region Constructor Region

        private Portal()
        {
        }

        public Portal(Point sourceTile, Point destinationTile, string destinationLevel)
        {
            SourceTile = sourceTile;
            DestinationTile = destinationTile;
            DestinationLevel = destinationLevel;
        }

        #endregion
    }
}
```

So, a portal had three properties in my game. It had a Point that held the X and Y coordinates of where the portal was located on the map. It had another Point that held the X and Y coordinates of the destination tile for the portal. It also had a string variable the held the name of the map/level that the portal led to.

I added three variables to the class to hold those three properties: sourceTile, destinationTile, and destinationLevel. I then included three properties that exposed their values but could not be set outside of the class. The reason for doing this is two fold. First, you typically do not want to change the value of a portal. In some instances you may want a portal to change where it is located or where it leads to but usually this is not the norm. I also made the set accessors private because there are required to serialize and deserialize maps. It is also why I included a private constructor that takes no parameters and has no actions in it. Finally, there is a constructor that takes as parameters, the source tile, the destination tile and the destination level. It then assigns those values to the fields in the class. You will also so that I marked all of the properties with the ContentSerializer attribute so that they are serialized and deserialized using the Intermediate Serializer.

Now that there is a class that represents a Portal I now added a layer to the map called PortalLayer. It was a collection of Portal objects on the map. Keep in mind when developing tile maps your layers

don't necessarily have to display graphics. They can also hold metadata about the map, in this case the portals on the map. Now, right click on the TileEngine folder, select Add and then Class. Name this new class PortalLayer.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;

namespace Avatars.TileEngine
{
    public class PortalLayer
    {
        #region Field Region

        private Dictionary<Rectangle, Portal> portals;

        #endregion

        #region Property Region

        [ContentSerializer]
        public Dictionary<Rectangle, Portal> Portals
        {
            get { return portals; }
            private set { portals = value; }
        }

        #endregion

        #region Constructor Region

        public PortalLayer()
        {
            portals = new Dictionary<Rectangle, Portal>();
        }

        #endregion
    }
}
```

So, what we have here is a single field called portals, a property that exposes it for public read access but private write access called Portals and a constructor that creates the field. It is also marked with the ContentSerializer attribute so that it will be serialized and deserialized by the Intermediate Serializer.

Now, the portal layer needs to be integrated into the existing TileMap class. The changes were made so that it would not break any of the current functionality. Open the TileMap class and update the code field, property and constructor regions.

```
        #region Field Region

        string mapName;
        TileLayer groundLayer;
        TileLayer edgeLayer;
        TileLayer buildingLayer;
        TileLayer decorationLayer;
```

```
        Dictionary<string, Point> characters;
        CharacterManager characterManager;
        PortalLayer portalLayer;

        [ContentSerializer]
        int mapWidth;

        [ContentSerializer]
        int mapHeight;

        TileSet tileSet;

        #endregion

        #region Property Region

        [ContentSerializer]
        public string MapName
        {
            get { return mapName; }
            private set { mapName = value; }
        }

        [ContentSerializer]
        public TileSet TileSet
        {
            get { return tileSet; }
            set { tileSet = value; }
        }

        [ContentSerializer]
        public TileLayer GroundLayer
        {
            get { return groundLayer; }
            set { groundLayer = value; }
        }

        [ContentSerializer]
        public TileLayer EdgeLayer
        {
            get { return edgeLayer; }
            set { edgeLayer = value; }
        }

        [ContentSerializer]
        public TileLayer BuildingLayer
        {
            get { return buildingLayer; }
            set { buildingLayer = value; }
        }

        [ContentSerializer]
        public PortalLayer PortalLayer
        {
            get { return portalLayer; }
            private set { portalLayer = value; }
        }

        [ContentSerializer]
        public Dictionary<string, Point> Characters
        {
            get { return characters; }
            private set { characters = value; }
```

```
        }

        public int MapWidth
        {
            get { return mapWidth; }
        }

        public int MapHeight
        {
            get { return mapHeight; }
        }

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

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

        #endregion

        #region Constructor Region

        private TileMap()
        {
        }

        private TileMap(TileSet tileSet, string mapName, PortalLayer portals = null)
        {
            this.characters = new Dictionary<string, Point>();
            this.tileSet = tileSet;
            this.mapName = mapName;
            characterManager = CharacterManager.Instance;

            portalLayer = portals != null ? portals : new PortalLayer();
        }

        public TileMap(
            TileSet tileSet,
            TileLayer groundLayer,
            TileLayer edgeLayer,
            TileLayer buildingLayer,
            TileLayer decorationLayer,
            string mapName,
            PortalLayer portalLayer = null)
            : this(tileSet, mapName, portalLayer)
        {
            this.groundLayer = groundLayer;
            this.edgeLayer = edgeLayer;
            this.buildingLayer = buildingLayer;
            this.decorationLayer = decorationLayer;

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

        #endregion
```

The change here is I added a field to hold the PortalLayer associated with the TileMap, a property to expose it externally as read and internally as write. I marked this with the ContentSeralizier property so that it will be serialized and deserialized. I add a PortalLayer parameter as an optional parameter that has the value of NULL. What this does is it makes it so that the change will not require you to find everywhere you create a TileMap object and add a new parameter to the call. In the second constructor I check to see if the portals parameter is not NULL. If it is not NULL I assign the value to the local variable, otherwise I create a new PortalLayer. If you know that an if statement is going to just set values by checking if an object has a value or not you can use the ? operator to do that. It is read **condition ? true action : false action**. It is a nice shorthand and cleans up the code a bit.

There is one other metadata class that I want to add. This class holds all of the maps for the game. I called this class World. Right click the TileEngine folder, select Add and then Class. Name this new class World. Here is the code for that class.

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

namespace Avatars.TileEngine
{
    public class World
    {
        #region Field Region

        private Dictionary<string, TileMap> maps;
        private string currentMapName;

        #endregion

        #region Property region

        [ContentSerializer]
        public Dictionary<string, TileMap> Maps
        {
            get { return maps; }
            private set { maps = value; }
        }

        [ContentSerializer]
        public string CurrentMapName
        {
            get { return currentMapName; }
            private set { currentMapName = value; }
        }

        public TileMap CurrentMap
        {
            get { return maps[currentMapName]; }
        }

        #endregion

        #region Constructor Region
```

```
        public World()
        {
            maps = new Dictionary<string, TileMap>();
        }

        #endregion

        #region Method Region

        public void AddMap(string mapName, TileMap map)
        {
            if (!maps.ContainsKey(mapName))
                maps.Add(mapName, map);
        }

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera)
        {
            CurrentMap.Draw(gameTime, spriteBatch, camera);
        }

        public void ChangeMap(string mapName, Rectangle portalLocation)
        {
            if (maps.ContainsKey(mapName))
            {
                currentMapName = mapName;
                return;
            }

            throw new Exception("Map name or portal name not found.");
        }

        #endregion
    }
}
```

This was a very simple class in my game that was responsible for managing all of the maps in the game. There were two fields in the class. One is a dictionary of the maps in the game and the other is a string that holds the name of the current map. There are properties marked with the ContentSerializer attribute so that the world can be serialized and deserialized. This allowed me to load all of the maps in my game at once. I also had an editor that worked with the maps. I will need to clean it up and update it a bit but I will eventually be posting the map editor that I used in this game. There are a few methods in this class. The one adds a map to the world, the second draws the current map.

There is also a method called ChangeMap. This is the method that will change the current map with the desired map, if it exists in the collection of maps. You probably should add a bit more validation here to make sure the destination is inside the map that you are switching to.

Now, this needs to be added to the GamePlayState to incorporate it into the existing game. I'm going to do this in stages because there are a lot of changes to the state to incorporate it into the game. First, in the GamePlayState, replace the TileMap map field with the field World world like this.

```
        Engine engine = new Engine(Game1.ScreenRectangle, 64, 64);
        World world;
        Camera camera;
        Player player;
```

Now, the Update method needs to be modified because it used the map field to search for characters

on the map. Change the Update method to the following.

```
        public override void Update(GameTime gameTime)
        {
            Vector2 motion = Vector2.Zero;
            int cp = 8;

            if (Xin.KeyboardState.IsKeyDown(Keys.W) && Xin.KeyboardState.IsKeyDown(Keys.A))
            {
                motion.X = -1;
                motion.Y = -1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.W) &&
Xin.KeyboardState.IsKeyDown(Keys.D))
            {
                motion.X = 1;
                motion.Y = -1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.S) &&
Xin.KeyboardState.IsKeyDown(Keys.A))
            {
                motion.X = -1;
                motion.Y = 1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.S) &&
Xin.KeyboardState.IsKeyDown(Keys.D))
            {
                motion.X = 1;
                motion.Y = 1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.W))
            {
                motion.Y = -1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkUp;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.S))
            {
                motion.Y = 1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkDown;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.A))
            {
                motion.X = -1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.D))
            {
                motion.X = 1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
            }

            if (motion != Vector2.Zero)
            {
                motion.Normalize();
                motion *= (player.Speed * (float)gameTime.ElapsedGameTime.TotalSeconds);

                Rectangle pRect = new Rectangle(
                    (int)player.Sprite.Position.X + (int)motion.X + cp,
```

```
                    (int)player.Sprite.Position.Y + (int)motion.Y + cp,
                    Engine.TileWidth - cp,
                    Engine.TileHeight - cp);

                foreach (string s in world.CurrentMap.Characters.Keys)
                {
                    ICharacter c = GameRef.CharacterManager.GetCharacter(s);
                    Rectangle r = new Rectangle(
                        (int)world.CurrentMap.Characters[s].X * Engine.TileWidth + cp,
                        (int)world.CurrentMap.Characters[s].Y * Engine.TileHeight + cp,
                        Engine.TileWidth - cp,
                        Engine.TileHeight - cp);

                    if (pRect.Intersects(r))
                    {
                        motion = Vector2.Zero;
                        break;
                    }
                }

                Vector2 newPosition = player.Sprite.Position + motion;

                player.Sprite.Position = newPosition;
                player.Sprite.IsAnimating = true;
                player.Sprite.LockToMap(new Point(world.CurrentMap.WidthInPixels,
world.CurrentMap.HeightInPixels));
            }
            else
            {
                player.Sprite.IsAnimating = false;
            }

            camera.LockToSprite(world.CurrentMap, player.Sprite, Game1.ScreenRectangle);
            player.Sprite.Update(gameTime);

            if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
            {
                foreach (string s in world.CurrentMap.Characters.Keys)
                {
                    ICharacter c = CharacterManager.Instance.GetCharacter(s);
                    float distance = Vector2.Distance(player.Sprite.Center, c.Sprite.Center);

                    if (Math.Abs(distance) < 72f)
                    {
                        IConversationState conversationState =
(IConversationState)GameRef.Services.GetService(typeof(IConversationState));
                        manager.PushState(
                            (ConversationState)conversationState,
                            PlayerIndexInControl);

                        conversationState.SetConversation(player, c);
                        conversationState.StartConversation();
                    }
                }
            }

            if (Xin.CheckKeyReleased(Keys.B))
            {
                foreach (string s in world.CurrentMap.Characters.Keys)
                {
                    ICharacter c = CharacterManager.Instance.GetCharacter(s);
                    float distance = Vector2.Distance(player.Sprite.Center, c.Sprite.Center);
```

```
                if (Math.Abs(distance) < 72f && !c.Battled)
                {
                    GameRef.BattleState.SetAvatars(player.CurrentAvatar, c.BattleAvatar);
                    manager.PushState(
                        (BattleState)GameRef.BattleState,
                         PlayerIndexInControl);
                    c.Battled = true;
                }
            }
        }
        base.Update(gameTime);
    }
```

All that I did in this method was replace all the instances of map with world.CurrentMap. I did the exact same thing in the Draw method. You can update it to the following code.

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

        if (world.CurrentMap != null && camera != null)
            world.CurrentMap.Draw(gameTime, GameRef.SpriteBatch, camera);

        GameRef.SpriteBatch.Begin(
            SpriteSortMode.Deferred,
            BlendState.AlphaBlend,
            SamplerState.PointClamp,
            null,
            null,
            null,
            camera.Transformation);

        player.Sprite.Draw(gameTime, GameRef.SpriteBatch);

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

The last change was to SetUpNewGame. In this method I initialized the world field, added a local variable to hold the map. I then created the map, added it to the world and changed the map to be this new map.

```
    public void SetUpNewGame()
    {
        Texture2D spriteSheet = content.Load<Texture2D>(@"PlayerSprites\maleplayer");
        TileMap map = null;
        world = new World();

        player = new Player(GameRef, "Wesley", false, spriteSheet);
        player.AddAvatar("fire", AvatarManager.GetAvatar("fire"));
        player.SetAvatar("fire");

        Texture2D tiles = GameRef.Content.Load<Texture2D>(@"Tiles\tileset1");
        TileSet set = new TileSet(8, 8, 32, 32);
        set.Texture = tiles;

        TileLayer background = new TileLayer(200, 200);
        TileLayer edge = new TileLayer(200, 200);
        TileLayer building = new TileLayer(200, 200);
        TileLayer decor = new TileLayer(200, 200);
```

```
            map = new TileMap(set, background, edge, building, decor, "test-map");

            map.FillEdges();
            map.FillBuilding();
            map.FillDecoration();

            ConversationManager.CreateConversations(GameRef);

            ICharacter teacherOne = Character.FromString(GameRef,
"Lance,teacherone,WalkDown,teacherone,water");
            ICharacter teacherTwo = PCharacter.FromString(GameRef,
"Marissa,teachertwo,WalkDown,tearchertwo,wind,earth");

            teacherOne.SetConversation("LanceHello");
            teacherTwo.SetConversation("MarissaHello");

            GameRef.CharacterManager.AddCharacter("teacherone", teacherOne);
            GameRef.CharacterManager.AddCharacter("teachertwo", teacherTwo);

            map.Characters.Add("teacherone", new Point(0, 4));
            map.Characters.Add("teachertwo", new Point(4, 0));

            map.PortalLayer.Portals.Add(Rectangle.Empty, new Portal(Point.Zero, Point.Zero,
"level1"));
            world.AddMap("level1", map);

            world.ChangeMap("level1", Rectangle.Empty);

            camera = new Camera();
        }
```

At this point I found a problem with the last tutorial. I missed implementing a member of the ICharacter interface for the PCharacter class so the game will not build. Add the following propery to the PCharacter class.

```
        public bool Battled
        {
            get
            {
                throw new NotImplementedException();
            }
            set
            {
                throw new NotImplementedException();
            }
        }
```

It will need to be fleshed out at some point but I'm going to do that in another tutorial. So, if you build and run the game you can start a new game and everything will work as expected. You can have a conversation with the NPCs and you can battle them. You still can't switch to another map. The first reason is that there is no portal to another map and second is that we haven't implemented that ability in the Update method. First, I will add the ability to switch maps in the Update method. Next I will update the SetUpNewGame method so that it creates two maps with a portal on each map that leads between the two maps.

I'm going to go with the default action, space bar or enter key, to activate an object, portal, initiate a conversation, etc. To do that I added the following changes to the Update method.

```
        public override void Update(GameTime gameTime)
        {
```

```
            Vector2 motion = Vector2.Zero;
            int cp = 8;

            if (Xin.KeyboardState.IsKeyDown(Keys.W) && Xin.KeyboardState.IsKeyDown(Keys.A))
            {
                motion.X = -1;
                motion.Y = -1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.W) &&
Xin.KeyboardState.IsKeyDown(Keys.D))
            {
                motion.X = 1;
                motion.Y = -1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.S) &&
Xin.KeyboardState.IsKeyDown(Keys.A))
            {
                motion.X = -1;
                motion.Y = 1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.S) &&
Xin.KeyboardState.IsKeyDown(Keys.D))
            {
                motion.X = 1;
                motion.Y = 1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.W))
            {
                motion.Y = -1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkUp;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.S))
            {
                motion.Y = 1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkDown;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.A))
            {
                motion.X = -1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.D))
            {
                motion.X = 1;
                player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
            }

            if (motion != Vector2.Zero)
            {
                motion.Normalize();
                motion *= (player.Speed * (float)gameTime.ElapsedGameTime.TotalSeconds);

                Rectangle pRect = new Rectangle(
                    (int)player.Sprite.Position.X + (int)motion.X + cp,
                    (int)player.Sprite.Position.Y + (int)motion.Y + cp,
                    Engine.TileWidth - cp,
                    Engine.TileHeight - cp);
```

```
                foreach (string s in world.CurrentMap.Characters.Keys)
                {
                    ICharacter c = GameRef.CharacterManager.GetCharacter(s);
                    Rectangle r = new Rectangle(
                        (int)world.CurrentMap.Characters[s].X * Engine.TileWidth + cp,
                        (int)world.CurrentMap.Characters[s].Y * Engine.TileHeight + cp,
                        Engine.TileWidth - cp,
                        Engine.TileHeight - cp);

                    if (pRect.Intersects(r))
                    {
                        motion = Vector2.Zero;
                        break;
                    }
                }

                Vector2 newPosition = player.Sprite.Position + motion;

                player.Sprite.Position = newPosition;
                player.Sprite.IsAnimating = true;
                player.Sprite.LockToMap(new Point(world.CurrentMap.WidthInPixels,
world.CurrentMap.HeightInPixels));
            }
            else
            {
                player.Sprite.IsAnimating = false;
            }

            camera.LockToSprite(world.CurrentMap, player.Sprite, Game1.ScreenRectangle);
            player.Sprite.Update(gameTime);

            if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
            {
                foreach (string s in world.CurrentMap.Characters.Keys)
                {
                    ICharacter c = CharacterManager.Instance.GetCharacter(s);
                    float distance = Vector2.Distance(player.Sprite.Center, c.Sprite.Center);

                    if (Math.Abs(distance) < 72f)
                    {
                        IConversationState conversationState =
(IConversationState)GameRef.Services.GetService(typeof(IConversationState));
                        manager.PushState(
                            (ConversationState)conversationState,
                             PlayerIndexInControl);

                        conversationState.SetConversation(player, c);
                        conversationState.StartConversation();
                    }
                }

                foreach (Rectangle r in world.CurrentMap.PortalLayer.Portals.Keys)
                {
                    Portal p = world.CurrentMap.PortalLayer.Portals[r];

                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        new Vector2(
                            r.X * Engine.TileWidth + Engine.TileWidth / 2,
                            r.Y * Engine.TileHeight + Engine.TileHeight / 2));

                    if (Math.Abs(distance) < 64f)
```

```
                    {
                        world.ChangeMap(p.DestinationLevel, new Rectangle(p.DestinationTile.X,
p.DestinationTile.Y, 32, 32));

                        player.Position = new Vector2(
                            p.DestinationTile.X * Engine.TileWidth,
                            p.DestinationTile.Y * Engine.TileHeight);
                        camera.LockToSprite(world.CurrentMap, player.Sprite,
Game1.ScreenRectangle);

                        return;
                    }
                }
            }

            if (Xin.CheckKeyReleased(Keys.B))
            {
                foreach (string s in world.CurrentMap.Characters.Keys)
                {
                    ICharacter c = CharacterManager.Instance.GetCharacter(s);
                    float distance = Vector2.Distance(player.Sprite.Center, c.Sprite.Center);

                    if (Math.Abs(distance) < 72f && !c.Battled)
                    {
                        GameRef.BattleState.SetAvatars(player.CurrentAvatar, c.BattleAvatar);
                        manager.PushState(
                            (BattleState)GameRef.BattleState,
                             PlayerIndexInControl);
                        c.Battled = true;
                    }
                }
            }
            base.Update(gameTime);
        }
```

Inside the if statement that checks if Space/Enter have been pressed I added another foreach loop that loops over the keys in the portal layer of the current map. I first get the portal using the key. I then calculate the distance between the character and the portal as I did for conversations and battling avatars. You will notice that I'm multiplying X and Y coordinates by TileWidth and TileHeight properties of the engine to get positions in pixels instead of tiles. If the portal and player are close enough together I change the map by calling ChangeMap passing in the name of the level and the destination rectangle. I then update the player's position to be the position on the new map. Then I lock the camera onto the player on the map and exit the Update method because there is no point in processing anything else in the method at that point.

The last thing that I'm going to cover in this tutorial is creating a basic second map with a portal that leads back to the same position as the first map. I did that in the SetUpNewGame method as that is where I already created a map and added it to the world. Change that map to the following.

```
public void SetUpNewGame()
{
    Texture2D spriteSheet = content.Load<Texture2D>(@"PlayerSprites\maleplayer");
    TileMap map = null;
    world = new World();

    player = new Player(GameRef, "Wesley", false, spriteSheet);
    player.AddAvatar("fire", AvatarManager.GetAvatar("fire"));
    player.SetAvatar("fire");
```

```
    Texture2D tiles = GameRef.Content.Load<Texture2D>(@"Tiles\tileset1");
    TileSet set = new TileSet(8, 8, 32, 32);
    set.Texture = tiles;

    TileLayer background = new TileLayer(200, 200);
    TileLayer edge = new TileLayer(200, 200);
    TileLayer building = new TileLayer(200, 200);
    TileLayer decor = new TileLayer(200, 200);

    map = new TileMap(set, background, edge, building, decor, "test-map");

    map.FillEdges();
    map.FillBuilding();
    map.FillDecoration();

    building.SetTile(4, 4, 18);

    ConversationManager.CreateConversations(GameRef);

    ICharacter teacherOne = Character.FromString(GameRef,
"Lance,teacherone,WalkDown,teacherone,water");
    ICharacter teacherTwo = PCharacter.FromString(GameRef,
"Marissa,teachertwo,WalkDown,tearchertwo,wind,earth");

    teacherOne.SetConversation("LanceHello");
    teacherTwo.SetConversation("MarissaHello");

    GameRef.CharacterManager.AddCharacter("teacherone", teacherOne);
    GameRef.CharacterManager.AddCharacter("teachertwo", teacherTwo);

    map.Characters.Add("teacherone", new Point(0, 4));
    map.Characters.Add("teachertwo", new Point(4, 0));

    map.PortalLayer.Portals.Add(Rectangle.Empty, new Portal(Point.Zero, Point.Zero, "level1"));
    map.PortalLayer.Portals.Add(new Rectangle(4, 4, 32, 32), new Portal(new Point(4, 4), new
Point(10, 10), "inside"));

    world.AddMap("level1", map);
    world.ChangeMap("level1", Rectangle.Empty);

    background = new TileLayer(20, 20, 23);
    edge = new TileLayer(20, 20);
    building = new TileLayer(20, 20);
    decor = new TileLayer(20, 20);

    map = new TileMap(set, background, edge, building, decor, "inside");
    map.FillEdges();
    map.FillBuilding();
    map.FillDecoration();
    map.BuildingLayer.SetTile(9, 19, 18);

    map.PortalLayer.Portals.Add(new Rectangle(9, 19, 32, 32), new Portal(new Point(9, 19), new
Point(4, 4), "level1"));

    world.AddMap("inside", map);

    camera = new Camera();
}
```

What changed as adding the map to the world is that I created a new ground layer filling it with tile 23 from the tile set we are using that is a floor like tile (close enough for me.) I then create the other layers at the same size. I then create the map, call the FillEdge, FillBuilding and FillDecoration methods. I then set one tile to be a door tile. I add a new portal that leads back to the original portal. I then add the map to world. I also set a tile on the first map to be the same door tile to give the visual cue that the player can open the door.

If you build and run the game now and go to the door and press Space/Enter you should now be able to switch to the new map. You can then walk to the far right of the map and switch back to the original map.

I'm going to end the tutorial here as I like to keep the tutorials to a reasonable length so there is not a lot of new code to digest at once. Please stay tuned for the next tutorial in this series. If you don't want to have to keep visiting the site to check for new tutorials you can sign up for my newsletter on the site and get a weekly status update of all the news from Game Programming Adventures. You can also follow my tutorials on Twitter at https://twitter.com/GPAAdmi77640534.

I wish you the best in your MonoGame Programming Adventures!
Cynthia McMahon