Eyes of the Dragon Tutorials 4.0

Part 15

Encounters

I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the Eyes of the Dragon 4.0 page of my web blog. I will be making the project available on GitHub here. It will be included on the page that links to the tutorials.

This is part fifteen of a series of tutorials I plan to write on creating a role-playing game with MonoGame. I've worked on similar tutorials using XNA and the past version of MonoGame. In the process of writing more tutorials, I discovered better ways of doing some things and had to go back to fix things. I'm hoping in this series to avoid making those same mistakes. Also, I am going to make the game cross-platform. In my previous tutorials, they were focused on Windows only. I want to open things up for macOS, Linux, Android, iOS, and any platform MonoGame supports.

Before I go much further in this tutorial, I want to update the Alive property of the encounter class. I want to check if a character has health greater than zero instead of checking the count of characters. Replace the Alive property of the Encounter class with this new version.

```
public bool Alive => Enemies.Any(x => x.Health.Current > 0) && Allies.Any(x =>
x.Health.Current > 0);
```

Instead of using the Count property, I use the Any method to check if the current health is greater than zero. Also, I want to change the Update method to do something similar. It also needs to call the Update method of the ally characters. Instead of Any, I use Where to return all characters that are alive. Replace the Update method with the following.

```
public void Update(GameTime gameTime)
{
    Map?.Update(gameTime);

    foreach (var character in Enemies.Where(x => x.Health.Current > 0))
    {
        character.Update(gameTime);
    }

    foreach (var character in Allies.Where(x => x.Health.Current > 0))
    {
        character.Update(gameTime);
    }
}
```

There is a tiny problem. The play can walk through the bat. We need to prevent them from doing that. That will be done in the TileMap class in the PlayerCollides method. Replace that method with this new version.

```
public bool PlayerCollides(Rectangle nextPotition)
{
    CharacterLayer layer = _mapLayers.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.Tile-
Height),
                new(Engine.TileWidth, Engine.TileHeight));
            if (rectangle.Intersects(nextPotition))
            {
                return true;
            }
        }
    }

    EncounterLayer encounter = _mapLayers.Where(x => x is EncounterLayer).FirstOrDe-
fault() as EncounterLayer;

    if (encounter != null)
    {
        foreach (var character in encounter.Encounters.Keys)
        {
            Rectangle rectangle = new(
                new((int)character.Position.X, (int)character.Position.Y),
                new(Engine.TileWidth, Engine.TileHeight));

            if (rectangle.Intersects(nextPotition))
            {
                return true;
            }
        }
    }
    return false;
}
```

Similar to the code where I get the character layer, I get the encounter layer using a bit of LINQ. If it is not null, I loop over the Encounters Keys property. I then create a rectangle that represents the position of the sprite. If that intersects the player's next position, I return true.

I envision three ways in which to start an encounter. First is that the player is beside an encounter and presses the E key. Second, they bump into the encounter. Third, the mob sees the player and initiates the encounter. I will demonstrate all three methods. The first method is familiar, and we implemented it with the conversations. To do that, replace the Update method of the gameplay state with the following.

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

    Player.Update(gameTime);
    map.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 (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.F) && !inMotion)
    {
        HandleConversation();
    }

    if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.E) && !inMotion)
    {
        HandleEncounter();
    }

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

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

    Vector2 newPosition = Player.Sprite.Position + motion * speed * (float)ga-
meTime.ElapsedGameTime.TotalSeconds;

    Rectangle nextPotition = new(
        (int)newPosition.X,
        (int)newPosition.Y,
        Engine.TileWidth,
        Engine.TileHeight);

    if (nextPotition.Intersects(collision))
    {
        inMotion = false;
        motion = Vector2.Zero;
```

```
        Player.Sprite.Position = new((int)Player.Sprite.Position.X,
(int)Player.Sprite.Position.Y);
        return;
    }

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

    Player.Sprite.Position = newPosition;
    Player.Sprite.Tile = Engine.VectorToCell(newPosition);

    camera.LockToSprite(Player.Sprite, map);

    base.Update(gameTime);
}
```

That leaves a new method, HandleEncounter, that will trigger a new encounter. Add this new method to the GamePlayState.

```
private void HandleEncounter()
{
    var layer = map.Layers.FirstOrDefault(x => x is EncounterLayer) as EncounterLayer;

    if (layer != null)
    {
        foreach (var c in layer.Encounters.Keys)
        {
            Point tile = Engine.VectorToCell(c.Position);

            if (tile.X == Player.Sprite.Tile.X && Math.Abs(Player.Sprite.Tile.Y - tile.Y)
== 1 ||
                (tile.Y == Player.Sprite.Tile.Y && Math.Abs(Player.Sprite.Tile.X -
tile.X) == 1))
            {
                _encounterState.SetEncounter(Player, layer.Encounters[c]);
                StateManager.PushState((EncounterState)_encounterState);
                break;
            }
        }
    }
}
```

First, we find the encounter layer using a little LINQ. If it is not null, we loop over the encounter's keys collection. We find what tile the encounter is in. Next, we check to see if the player is adjacent to the player, the same way we did for conversations. Next, we call a method SetEncounter on the new encounter state to set the encounter. Next, we push the encounter state onto the stack and break out of the loop.

I just added a new state, EncounterState, that will run encounters. Instead of throwing a whole wack of code at you, I will add it piece by piece. Right-click the GameScreens folder in the SharedProject project, select Add and then Class. Name this new class EncounterState. Here is the initial code for the

class.

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

namespace SharedProject.GamesScreens
{
    public interface IEncounterState
    {
        void SetEncounter(Player player, Encounter encounter);
    }

    public class EncounterState : GameState, IEncounterState
    {
        private Encounter encounter;
        private Camera Camera { get; set; }
        private RenderTarget2D renderTarget;

        public EncounterState(Game game) : base(game)
        {
            Game.Services.AddService(typeof(IEncounterState), this);
            Camera = new(new(Point.Zero, new(Settings.BaseWidth, Settings.BaseHeight)));
        }

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

            renderTarget = new(GraphicsDevice, Settings.BaseWidth, Settings.BaseHeight);
        }
        public void SetEncounter(Player player, Encounter encounter)
        {
            this.encounter = encounter;
        }

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

            encounter?.Update(gameTime);
        }

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

            GraphicsDevice.SetRenderTarget(renderTarget);
            GraphicsDevice.Clear(Color.Black);

            SpriteBatch.Begin();

            Encounter?.Draw(gameTime, SpriteBatch, Camera);

            SpriteBatch.End();
```

```
            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin();
            SpriteBatch.Draw(renderTarget, Vector2.Zero, Color.White);
            SpriteBatch.End();
        }
    }

}
```

As with all states, there is an interface that the state will implement. It has a single method, SetEncounter, that we used in the GamePlayState that will start an encounter. It requires as arguments the Player object and an Encounter. The class inherits from GameState and implements the IEncounterState interface.

There are currently three fields. An Encounter, Camera, and RenderTarget2D. The first is the encounter that the player is in. The next is the camera that will be used to render the encounter because it uses a tile map. The last field is the render target that is used to handle multiple resolutions.

The constructor registers the state as a service that can be retrieved in other classes. Next, it creates a camera. It uses a rectangle that has its X and Y coordinates as (0, 0) and a height and width of the base resolution. The LoadContent method created the render target for rendering in multiple resolutions. The Update method calls base.Update and the Update method of the encounter, if it is not null. In the Draw method, things flow the same as in other states. First is the call on the Draw method of the base class. Next, set the render target and clear the render target. It calls Begin to start rendering, draws the encounter if is not null, then stops rendering. It resets the render target and then draws the scaled render target.

If you build and run now, you can walk up to the bat and start an encounter. There are a few problems. First, the player and bat are still at their positions and animation on the map. Second, there is no map. How are we going to tackle these problems? Let's start with position and animations first. What we need to do is capture the player and mob's position and current animation when combat begins. Next, we need to change it to the starting position on the map. Finally, when the encounter is done, we need to reset the player's position and animation. So, to do that, we will change the SetEncounter method. Add the following properties to the EncounterState class and update the SetEncounter method to the following.

```
public Point PlayerTile { get; private set; }
public Vector2 PlayerPosition { get; private set; }
public string PlayerAnimation { get; private set; }

public void SetEncounter(Player player, Encounter encounter)
{
    this.encounter = encounter;

    PlayerTile = player.Tile;
    PlayerPosition = player.Position;
    PlayerAnimation = player.Sprite.CurrentAnimation;

    Point newTile = new(3, 5);
```

```
    this.encounter.Allies[0].Tile = newTile;
    ((Player)this.encounter.Allies[0]).Sprite.Position = new(newTile.X * Engine.Tile-
Width, newTile.Y * Engine.TileHeight);
    ((Player)this.encounter.Allies[0]).Sprite.CurrentAnimation = "walkright";
    ((Player)this.encounter.Allies[0]).Sprite.IsAnimating = true;

    ((Mob)this.encounter.Enemies[0]).AnimatedSprite.Position = new(16 * Engine.TileWidth,
5 * Engine.TileHeight);
    ((Mob)this.encounter.Enemies[0]).AnimatedSprite.CurrentAnimation = "left";
}
```

What happens here is that after setting the encounter, I grab the player's current tile, position and animation. I then create a new point that the player will start in. I set the player's tile to the new tile. I set the position of the sprite based on the new tile and the engine's tile width and tile height properties. I set the current animation to walk right, and is animating to true. Ideally, I would set the current animation to idle. However, I do not have an idle animation for these sprites. I then set the position and current animation of the mob to a new tile on the right side of the map using the tile width and tile height properties of the engine.

Before building and running, I want to change the tile width and height in the game to 64 by 64 pixels. It just "looks better" in the game and in an encounter. First, replace the constructor of the GamePlayState with this new version.

```
public GamePlayState(Game game) : base(game)
{
    Game.Services.AddService<IGamePlayState>(this);
    camera = new(Settings.BaseRectangle);
    engine = new(64, 64, Settings.BaseRectangle);
}
```

Because we've changed the pixel density, the sprite will move painfully slowly. We can change that in the Initialize method. All we need to do is increase the speed. Replace the initialize method with the following version.

```
public GamePlayState(Game game) : base(game)
{
    Game.Services.AddService<IGamePlayState>(this);
    camera = new(Settings.BaseRectangle);
    engine = new(64, 64, Settings.BaseRectangle);
}
```

So, I need to make some adjustments to creating the game. What I need to do is change the position of the villager, and update the string for creating the bat. The bat's animations are dead animation in the first column and fly animations in the last three. So, I'm going to add dead animations and change the X offset when creating the fly animations to start at the second column instead of the first column. Replace the LoadContent method of the GamePlayState with this new version.

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

    _conversationState = Game.Services.GetService<IConversationState>();
```

```
    _conversationManager = Game.Services.GetService<IConversationManager>();
    _encounterState = Game.Services.GetService<IEncounterState>();

    renderTarget = new(GraphicsDevice, Settings.BaseWidth, Settings.BaseHeight);

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

    List<Tileset> tilesets = new()
    {
        new(texture, 8, 8, 32, 32),
    };

    TileLayer layer = new(100, 100);

    map = new("test", tilesets[0], layer);

    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/femalefighter");

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

    CharacterLayer chars = new();

    chars.Characters.Add(
        new Villager(rio, new(10, 10))
        {
            Position = new(480, 480),
            Tile = new(10, 10),
            Visible= true,
            Enabled=true,
            Conversation="Rio"
        });

    map.AddLayer(chars);

    EncounterLayer encounters = new();

    Encounter encounter = new(Player);
    encounter.Enemies.Add(Mob.FromString("Name=Giant Bat,Strength=3,Agil-
ity=5,Health=21,Position=640:640,Tile=5:5,AnimatedSprite=32x32-bat-
sprite;down:32:0:32:32:3;right:32:32:32:32:3;up:32:64:32:32:3;left:32:96:32:32:3;deaddown
:0:0:32:32:1;deadright:0:32:32:32:1;deadup:0:64:32:32:1;deadleft:0:96:32:32:1;down",
Game.Content));
```

```
    encounters.Encounters.Add(((Mob)encounter.Enemies[0]).AnimatedSprite, encounter);
    map.AddLayer(encounters);

    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;
    ControlManager.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;
    ControlManager.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;
    ControlManager.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;

    ControlManager.Add(leftButton);
}
```

If you build and run now, you can approach the bat and start combat. When you get to the combat screen, you and the bat will be at opposite ends of the field. We still have a black screen, though. Definitely, something we need to change. For now, I will create a random map on the fly. Eventually, we will use a map editor to generate maps. It will offer a random map option that can be customized. So, change the SetEncounter method to the following version.

```
public void SetEncounter(Player player, Encounter encounter)
{
```

```
    this.encounter = encounter;

    encounter.RandomMap(Game.Content);

    PlayerTile = player.Tile;
    PlayerPosition = player.Position;
    PlayerAnimation = player.Sprite.CurrentAnimation;

    Point newTile = new(3, 5);

    this.encounter.Allies[0].Tile = newTile;
    ((Player)this.encounter.Allies[0]).Sprite.Position = new(newTile.X * Engine.Tile-
Width, newTile.Y * Engine.TileHeight);
    ((Player)this.encounter.Allies[0]).Sprite.CurrentAnimation = "walkright";
    ((Player)this.encounter.Allies[0]).Sprite.IsAnimating = true;

    newTile = new(16, 5);

    this.encounter.Enemies[0].Tile = newTile;
    ((Mob)this.encounter.Enemies[0]).AnimatedSprite.Position = new(16 * Engine.TileWidth,
5 * Engine.TileHeight);
    ((Mob)this.encounter.Enemies[0]).AnimatedSprite.CurrentAnimation = "left";
}
```

All it does is call a new method, RandomMap, on the Encounter class. Add this method to the Encounter class. I also added a tile to the enemy. The reason while will become apparent soon.

```
public void RandomMap(ContentManager content)
{
    Random random = new();

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

    List<Tileset> tilesets = new()
    {
        new(texture, 8, 8, 32, 32),
    };

    TileLayer layer = new(1280 / 64 + 1, 720 / 64 + 1);

    Map = new("test", tilesets[0], layer);

    layer = new(1280 / 64 + 1, 720 / 64 + 1);

    for (int y = 0; y < 720 / 64 + 1; y++)
        for (int x = 0; x < 1280 / 64 + 1; x++)
        {
            layer.SetTile(x, y, new Tile(-1, -1));
        }

    for (int  i = 0; i < 100; i++)
    {
        layer.SetTile(random.Next(1 + 1280 / 64),
                      random.Next(1 + 720 / 64),
                      new Tile(random.Next(2, 14), 0));
    }

    Map.Layers.Add(layer);
}
```

Sadly, we do not have access to the Settings class in this project. That is because the SharedProject depends upon it, and adding a reference to the SharedProject creates a circular dependency. I will fix access to the Settings class in a future tutorial. For that reason, you will see 1280 and 720 throughout the class.

The method takes as a parameter the ContentManager, so that it can load in the texture for the tile set. It has a local Random variable to generate some splatter. It would probably be more efficient to generate it at the class level. This worked, though, so I'm sticking with it. By splatter, I mean some random tiles that will make it so that it is not plain green canvas. I then use the content manager to load the tile set. Following that, the method flows like creating our test map. However, instead of ending at a single layer, I create the splatter layer that I spoke about earlier. It is a layer filled with mainly empty tiles. However, I add one hundred random tiles from the tile set between tile 2, inclusive, and tile 14, exclusive. I place it on a random tile away from the edges of the map. The layer is then added to the list of layers on the map.

If you build and run now, you can walk up to the bat and initiate combat by pressing the F key. There you and the bat will be at opposite ends of the field with some obstacles between you. There are four problems. The major one is that there is tearing when rendering the second layer. Secondly, you cannot get any closer to the bat in order to fight. Third, the objects are pretty but useless. Finally, obstacles can appear underneath sprites. Let's tackle the first one. Update the draw method of the encounter state to the following.

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

    GraphicsDevice.SetRenderTarget(renderTarget);
    GraphicsDevice.Clear(Color.Black);

    SpriteBatch.Begin(SpriteSortMode.Deferred,
                      BlendState.AlphaBlend,
                      SamplerState.PointWrap);

    encounter?.Draw(gameTime, SpriteBatch, Camera);

    SpriteBatch.End();

    GraphicsDevice.SetRenderTarget(null);

    SpriteBatch.Begin();
    SpriteBatch.Draw(renderTarget, Vector2.Zero, Color.White);
    SpriteBatch.End();
}
```

What has changed here is the call to Begin on the SpriteBatch object. I set the sprite sort mode to deferred to have the rendering done when the call to Begin is finished. Next, I pass in a blend state of AlphaBlend because we will render sprites that have transparent parts. Finally, I pass in PointWrap as the sampler state. It is this that is preventing the tearing that we saw. If we were rendering on partial pixels, PointClamp is the better option.

Now, I will tackle obstacles appearing behind the player and the enemy. This is why I added the tile to the enemy in the encounter state. We need to check if the tile is appearing under enemies. I say enemies, and allies for that matter, because eventually, there may be more than one. We will do single entities for now. Replace the RandomMap method of the Encounter class with this new version.

```
public void RandomMap(ContentManager content)
{
    Random random = new();

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

    List<Tileset> tilesets = new()
    {
        new(texture, 8, 8, 32, 32),
    };

    TileLayer layer = new(1280 / 64 + 1, 720 / 64 + 1);

    Map = new("test", tilesets[0], layer);

    layer = new(1280 / 64 + 1, 720 / 64 + 1);

    for (int y = 0; y < 720 / 64 + 1; y++)
        for (int x = 0; x < 1280 / 64 + 1; x++)
        {
            layer.SetTile(x, y, new Tile(-1, -1));
        }

    for (int i = 0; i < 20; i++)
    {
        int x;
        int y;

        do
        {
            x = random.Next(1 + 1280 / 64);
            y = random.Next(1 + 720 / 64);
        } while (Allies.Any(z => z.Tile.X == x && z.Tile.Y == y) ||
                    Enemies.Any(z => z.Tile.X == x && z.Tile.Y == y));

        layer.SetTile(x, y, new Tile(random.Next(3, 14), 0));
    }

    Map.Layers.Add(layer);
}
```

So, what had changed here? Well, I introduced so local variables inside the loop that holds the X and Y coordinates that we want to place the tile. Next, there is a do-while loop. Inside the loop, I generate the x and y coordinates that we to place the obstacle. Then, in the while clause, I use a little LINQ. I check to see if there are any allies or enemies at the chosen tile.

Two problems down, two more to go. Those are that you can't move and obstacles are pretty but useless. Well, there is a third, and that is that the bat does not move. I think I will save that for a future tutorial. For movement, we will borrow from the GamePlayState to keep movement tile-based. I won't

be implementing the buttons, as we did for Android and iOS. I will circle back for Android and iOS later, in tutorials devoted specifically to those platforms. First, add the fields required for movement.

```
        private Rectangle collision;
        private bool inMotion;
        private Vector2 motion;
        private readonly float speed = 160;
```

Those are copied and passed from the GamePlayState. There is the collision rectangle to restrict movement to one tile at a time. The inMotion field tells if the player's sprite is already in motion. The motion field describes the direction to move the sprite. Finally, the speed field is how fast the sprite will move. Now, replace the Update method of the EncounterState with the following code, and add these four new movement methods.

```
private void MoveLeft()
{
    motion = new(-1, 0);
    inMotion = true;
    Player.Sprite.CurrentAnimation = "walkleft";
    collision = new(
        (Player.Sprite.Tile.X - 2) * Engine.TileWidth,
        Player.Sprite.Tile.Y * Engine.TileHeight,
        Engine.TileWidth,
        Engine.TileHeight);
}

private void MoveRight()
{
    motion = new(1, 0);
    inMotion = true;
    Player.Sprite.CurrentAnimation = "walkright";
    collision = new(
        (Player.Sprite.Tile.X + 2) * Engine.TileWidth,
        Player.Sprite.Tile.Y * Engine.TileHeight,
        Engine.TileWidth,
        Engine.TileHeight);
}

private void MoveDown()
{
    motion = new(0, 1);
    Point newTile = Player.Sprite.Tile + new Point(0, 2);
    inMotion = true;
    Player.Sprite.CurrentAnimation = "walkdown";
    collision = new(
        newTile.X * Engine.TileWidth,
        newTile.Y * Engine.TileHeight,
        Engine.TileWidth,
        Engine.TileHeight);
}

private void MoveUp()
{
    motion = new(0, -1);
    inMotion = true;
    Player.Sprite.CurrentAnimation = "walkup";
    collision = new(
```

```
        Player.Sprite.Tile.X * Engine.TileWidth,
        (Player.Sprite.Tile.Y - 2) * Engine.TileHeight,
        Engine.TileWidth,
        Engine.TileHeight);
}

public override void Update(GameTime gameTime)
{
    ControlManager.Update(gameTime);

    Player.Update(gameTime);
    encounter.Map.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();
        Player.Sprite.IsAnimating = true;
    }
    else
    {
        inMotion = false;
        Player.Sprite.IsAnimating = false;
        return;
    }

    if (!Player.Sprite.LockToMap(new(20 * Engine.TileWidth, 10 * Engine.TileHeight), ref
motion))
    {
        inMotion = false;
        return;
    }

    Vector2 newPosition = Player.Sprite.Position + motion * speed * (float)ga-
meTime.ElapsedGameTime.TotalSeconds;

    Rectangle nextPotition = new(
        (int)newPosition.X,
        (int)newPosition.Y,
        Engine.TileWidth,
        Engine.TileHeight);

    if (nextPotition.Intersects(collision))
```

```
    {
        inMotion = false;
        motion = Vector2.Zero;
        Player.Sprite.Position = new((int)Player.Sprite.Position.X,
(int)Player.Sprite.Position.Y);
        return;
    }

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

    Player.Sprite.Position = newPosition;
    Player.Sprite.Tile = Engine.VectorToCell(newPosition);

    base.Update(gameTime);

    encounter?.Update(gameTime);
}
```

This should all be familiar. It is the same code from the GamePlayState for moving the player's sprite. I will gloss over one of the four movements, leaving the other three for your review. Then, I will go over the changes to the Update method. In the Move* methods, the first step is to set the motion vector to the direction to move. Next, set inMotion to true. Following that, set the current animation. Finally, create a collision rectangle two tiles away.

The Update method is also pretty much a copy and paste of the Update method from the GamePlayState. The first step is to call the Update method of the control manager. Following that, it calls the Update method of the Player property, which hasn't been added yet. There is also a call to the Update method of the map for the encounter. Following that, are the checks to see if a key is down and that the sprite is not in motion. If those conditions are true, the move method is called. If the motion vector is not the zero vector, I normalize it and set the IsAnimating property of the player's sprite is set to true. Otherwise, IsAnimating is set to false, inMotion is set to false. Then, I lock the player's sprite to the map. I get the next position of the player's sprite using the current position and the motion of the sprite. I create a rectangle that describes it. If that intersects with the collision rectangle, I cancel the movement. Otherwise, I update the position. Next, I call a method that won't do anything right now. That is because the map does not have a character layer or encounter layer. I set the player's sprite's position to the new position. I update the tile. Finally, I call base.Update and the Update method of the encounter.

If you build and run now, and trigger the encounter, you can move the player's sprite around the map. There is the problem that you can walk over obstacles and the mob. Not cool! Let's fix the obstacles. To do that, we need to add a new layer to maps, a collision layer. This layer will have rectangles that, if the player intersects with one, movement will be cancelled. Then, in the PlayerCollides method, we will return true if there is an intersection. First, let's add the collision layer. Right-click the TileEngine folder in Psilibrary, select Add and the Class. Name this new class CollisionLayer. There is the code.

```
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 RpgLibrary.TileEngine
{
    public enum CollisionValue { Impassible, Water }

    public class CollisionLayer : ILayer
    {
        public Dictionary<Rectangle, CollisionValue> Collisions { get; private set; } =
new();

        public void Draw(SpriteBatch spriteBatch, Camera camera, List<Tileset> tilesets)
        {
        }

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

First, there is an enumeration to state what type of collision it is. While I most likely will not be implementing water, I included it as an option. It implements the ILayer interface so it can easily be added to the map as a layer. For the collisions, I added a property that is a Dictionary<Rectangle, CollisionsValue>. If there is a collision with that rectangle, some action will happen. Currently, we will only cancel the movement. There are also Draw and Update methods that are part of the ILayer interface that must be implemented, even if they do nothing.

Now, we need to update the RandomMap method of the encounter class to generate collisions. Update the RandomMap class to the following.

```
public void RandomMap(ContentManager content)
{
    Random random = new();

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

    List<Tileset> tilesets = new()
    {
        new(texture, 8, 8, 32, 32),
    };

    TileLayer layer = new(1280 / 64 + 1, 720 / 64 + 1);

    Map = new("test", tilesets[0], layer);

    layer = new(1280 / 64 + 1, 720 / 64 + 1);
    CollisionLayer collisions = new();

    for (int y = 0; y < 720 / 64 + 1; y++)
        for (int x = 0; x < 1280 / 64 + 1; x++)
```

```
        {
            layer.SetTile(x, y, new Tile(-1, -1));
        }

    for (int i = 0; i < 20; i++)
    {
        int x;
        int y;

        do
        {
            x = random.Next(1 + 1280 / 64);
            y = random.Next(1 + 720 / 64);
        } while (Allies.Any(z => z.Tile.X == x && z.Tile.Y == y) ||
                    Enemies.Any(z => z.Tile.X == x && z.Tile.Y == y));

        collisions.Collisions.Add(new(
            new(x * Engine.TileWidth, y * Engine.TileHeight),
            new(Engine.TileWidth, Engine.TileHeight)),
            CollisionValue.Impassible);

        layer.SetTile(x, y, new Tile(random.Next(3, 14), 0));
    }

    Map.Layers.Add(layer);
    Map.Layers.Add(collisions);
}
```

What I did was add a new local variable, collisions, that is a collision layer. In the loop that is creating the collisions, I add a new collision using the x and y coordinates and the tile width and tile height. I set it to Impassible. Then, like with the layer I add it to the map.

That leaves the PlayerCollides method to handle collisions. Replace the PlayerCollides method of the TileMap class with this new version.

```
public bool PlayerCollides(Rectangle nextPotition)
{
    if (_mapLayers.FirstOrDefault(x => x is CharacterLayer) is CharacterLayer layer)
    {
        foreach (var character in layer.Characters)
        {
            Rectangle rectangle = new(
                new(character.Tile.X * Engine.TileWidth, character.Tile.Y * Engine.Tile-
Height),
                new(Engine.TileWidth, Engine.TileHeight));
            if (rectangle.Intersects(nextPotition))
            {
                return true;
            }
        }
    }

    if (_mapLayers.FirstOrDefault(x => x is EncounterLayer) is EncounterLayer encounter)
    {
        foreach (var character in encounter.Encounters.Keys)
        {
            Rectangle rectangle = new(
                new((int)character.Position.X, (int)character.Position.Y),
```

```
                new(Engine.TileWidth, Engine.TileHeight));

            if (rectangle.Intersects(nextPotition))
            {
                return true;
            }
        }
    }

    if (_mapLayers.FirstOrDefault(x => x is CollisionLayer) is CollisionLayer collisions)
    {
        foreach (var r in collisions.Collisions.Keys)
        {
            if (r.Intersects(nextPotition) && collisions.Collisions[r] == Collision-
Value.Impassible)
            {
                return true;
            }
        }
    }
    return false;
}
```

As you can see, this is a new and improved version of the checks for collisions. Instead of having a separate call to get the layer and check if it is null, I am using pattern matching. Also, the LINQ has been shortened. Instead of using Where and FirstOrDefault, I just use FirstOrDefault.

What the new code does is grab the collision layer. Then, it loops over all of the collisions. It checks if the collision intersects with the next position and if the collision type is Impassible. If those two conditions are true, it returns true.

If you build and run now, you can initiate combat with the bat by walking up to it and pressing the E key. You can walk around the map but not over obstacles. That is pretty amazing for one tutorial. Look for the next tutorial over the weekend.

I'm going to park this tutorial here. There is more that I could pack in, but I will save it for the following tutorials. I don't want you to have too much to digest at once. I encourage you to visit the news page of my site, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials. Also, I'm thinking of reviving my newsletter so you will be informed of new stuff rather than having to keep looking for new things.

Good luck with your game programming adventures!
Cynthia