Eyes of the Dragon Tutorials 4.0

Part 16

Encounters- Part Two

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 sixteen 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.

Picking up from last time, we were working on encounters with mobs. I made some pretty significant changes to the EncounterState. So much that I'm going to give you the code for the entire class and then go over the code. I haven't added intelligence to mobs yet. That is just a single method that I will handle separately. Replace the EncounterState with the following code.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RpgLibrary;
using RpgLibrary.Characters;
using RpgLibrary.TileEngine;
using SharedProject.Controls;
using SharedProject.Mobs;
using SharedProject.Sprites;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.ExceptionServices;
using System.Text;

namespace SharedProject.GamesScreens
{
    public enum Mode { Movement, Attack, Cast, Use }

    public interface IEncounterState
    {
        void SetEncounter(Player player, Encounter encounter);
    }

    public class EncounterState : GameState, IEncounterState
    {
        private Mode _mode = Mode.Movement;
```

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

        private Encounter encounter;
        private Camera Camera { get; set; }
        private RenderTarget2D renderTarget;

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

        private bool _turn;
        private double _timer;
        private readonly Queue<string> _messages = new();
        private Texture2D _messageBox;

        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);
            _messageBox = new(GraphicsDevice, Settings.BaseWidth, 128);
            _messageBox.Fill(Color.White);
        }

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

            encounter.RandomMap(Game.Content);

            PlayerTile = player.Sprite.Tile;
            PlayerPosition = player.Sprite.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 * En-
gine.TileWidth, 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 * En-
gine.TileWidth, 5 * Engine.TileHeight);
            ((Mob)this.encounter.Enemies[0]).AnimatedSprite.CurrentAnimation = "left";

            Player = player;
```

```
        }

        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);
            if (encounter.Map.Layers.FirstOrDefault(x => x is CollisionLayer) is Colli-
sionLayer layer)
            {
                Dictionary<Rectangle, CollisionValue> collisions = layer.Collisions;
                if (!collisions.ContainsKey(new(Player.Sprite.Tile + new Point(-1, 0),
new(Engine.TileWidth, Engine.TileHeight))))
                {
                    Player.Tile += new Point(-1, 0);
                }
            }
        }

        private void MoveRight()
        {
            motion = new(1, 0);
            if (encounter.Map.Layers.FirstOrDefault(x => x is CollisionLayer) is Colli-
sionLayer layer)
            {
                Dictionary<Rectangle, CollisionValue> collisions = layer.Collisions;
                if (!collisions.ContainsKey(new(Player.Sprite.Tile + new Point(1, 0),
new(Engine.TileWidth, Engine.TileHeight))))
                {
                    Player.Tile += new Point(1, 0);
                }
            }
            Point newTile = Player.Sprite.Tile + new Point(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, 1);
            inMotion = true;
            Player.Sprite.CurrentAnimation = "walkdown";
            collision = new(
                newTile.X * Engine.TileWidth,
                (newTile.Y +  1) * Engine.TileHeight,
                Engine.TileWidth,
                Engine.TileHeight);
            if (encounter.Map.Layers.FirstOrDefault(x => x is CollisionLayer) is Colli-
sionLayer layer)
```

```
            {
                Dictionary<Rectangle, CollisionValue> collisions = layer.Collisions;
                if (!collisions.ContainsKey(new(Player.Sprite.Tile + new Point(1, 0),
new(Engine.TileWidth, Engine.TileHeight))))
                {
                    Player.Tile += new Point(0, 1);
                }
            }
        }

        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);
            if (encounter.Map.Layers.FirstOrDefault(x => x is CollisionLayer) is Colli-
sionLayer layer)
            {
                Dictionary<Rectangle, CollisionValue> collisions = layer.Collisions;
                if (!collisions.ContainsKey(new(Player.Sprite.Tile + new Point(1, 0),
new(Engine.TileWidth, Engine.TileHeight))))
                {
                    Player.Tile += new Point(0, -1);
                }
            }
        }

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

            encounter?.Update(gameTime);

            Player.Update(gameTime);
            encounter.Map.Update(gameTime);

            if (!encounter.Alive)
            {
                Player.Sprite.Tile = PlayerTile;
                Player.Sprite.Position = PlayerPosition;
                Player.Sprite.CurrentAnimation = PlayerAnimation;

                StateManager.PopState();
            }

            _timer += gameTime.ElapsedGameTime.TotalSeconds;

            if (_turn && _timer > 0.25)
            {
                if (_mode == Mode.Attack)
                {
                    HandleAttack();
                    return;
                }
```

```
                if (_mode == Mode.Movement)
                {
                    if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.F))
                    {
                        _mode = Mode.Attack;
                        _messages.Enqueue("Attack where...");
                        return;
                    }
                }
                if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.A) && !inMotion)
                {
                    MoveLeft();
                    _turn = false;
                    _timer = 0;
                    _messages.Enqueue("Moving west...");
                }
                else if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.D) && !inMo-
tion)
                {
                    MoveRight();
                    _turn = false;
                    _timer = 0;
                    _messages.Enqueue("Moving east...");
                }

                if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.W) && !inMotion)
                {
                    MoveUp();
                    _turn = false;
                    _timer = 0;
                    _messages.Enqueue("Moving north...");
                }
                else if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.S) && !inMo-
tion)
                {
                    MoveDown();
                    _turn = false;
                    _timer = 0;
                    _messages.Enqueue("Moving south...");
                }

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

            }
            else if (!_turn && _timer > 0.25)
            {
                Player.Sprite.IsAnimating = false;

                HandleEnemies();
            }
```

```
            if (!Player.Sprite.LockToMap(new(19 * Engine.TileWidth, 7 * Engine.Tile-
Height), 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);
                Player.Sprite.IsAnimating = false;

                return;
            }

            if (encounter.Map.PlayerCollides(nextPotition.Grow(-1)))
            {
                _messages.Enqueue("Ouch!");
                inMotion = false;
                motion = Vector2.Zero;
                return;
            }

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

            base.Update(gameTime);
        }

        private void HandleAttack()
        {
            if (_timer > 0.5)
            {
                if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.A))
                {
                    _timer = 0;
                    _turn = false;
                    _messages.Enqueue("   Attacking west...");
                    DoAttack(new(-1, 0));
                    _mode = Mode.Movement;
                }
                else if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.D))
                {
                    _timer = 0;
                    _turn = false;
                    _messages.Enqueue("   Attacking east...");
```

```
                    DoAttack(new(1, 0));
                    _mode = Mode.Movement;
                }
                else if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.W))
                {
                    _timer = 0;
                    _turn = false;
                    _messages.Enqueue("   Attacking north...");
                    DoAttack(new(0, -1));
                    _mode = Mode.Movement;
                }
                else if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.S))
                {
                    _timer = 0;
                    _turn = false;
                    _messages.Enqueue("   Attacking south...");
                    DoAttack(new(0, 1));
                    _mode = Mode.Movement;
                }
            }
        }

        private void DoAttack(Point direction)
        {
            Point target = Player.Tile + direction;
            Rectangle destination = new(target, new(Engine.TileWidth, Engine.Tile-
Height));

            for (int i =  0; i < encounter.Enemies.Count; i++)
            {
                var enemy = encounter.Enemies[i];
                Point enemyTile = new((int)((Mob)enemy).AnimatedSprite.Position.X / En-
gine.TileWidth,
                    (int)((Mob)enemy).AnimatedSprite.Position.Y / Engine.TileHeight);
                Rectangle enemyDestination = new(enemyTile, new(Engine.TileWidth, En-
gine.TileHeight));
                if (enemyDestination.Intersects(destination))
                {
                    _messages.Enqueue("   Enemy was hit...");

                    AttributePair health = new(enemy.Health.Maximum);
                    health.Current -= enemy.Health.Maximum;

                    enemy.Health = health;
                }
                else
                {
                    _messages.Enqueue("   Your attack fell upon empty air...");
                }
            }
        }

        private void HandleEnemies()
        {
            _turn = !_turn;
            _timer = 0;
        }

        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);

            Point messageLocation = new(0, Settings.BaseHeight - _messageBox.Height + 5);

            SpriteBatch.Draw(_messageBox,
                             messageLocation,
                             Color.White);
            messageLocation.X += 10;
            if (_messages.Count > 4)
            {
                _messages.Dequeue();
            }

            foreach (string s in _messages)
            {
                SpriteBatch.DrawString(ControlManager.SpriteFont, s, messageLocation,
Color.Black);
                messageLocation.Y += ControlManager.SpriteFont.LineSpacing;
            }

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

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

        protected override void Show()
        {
            _turn = false;
            _timer = 0;
            _messages.Clear();

            base.Show();
        }
    }

}
```

So, radically different from the previous version. There is an enumeration that is the state for input. There is movement, attack, cast and use. They are what you would suggest they are. Movement is for moving, Attack is for attacking, Cast is for using a skill, and Use is for using an item. There is a new field _mode that is the mode the state is in. As well, there is a field for who's turn it is and a field for the time passed between turns.

There are two other fields that are related to each other. The first is a Queue that will hold messages

to be displayed to the player. The second is a Texture2D that will be the message background. In the LoadContent method, I create a texture the width of the screen and 128 pixels tall.

There were a few minor changes to the SetEncounter method. Instead of using the Tile and Position properties of the Player object for saving the tile and position, I use the Tile and Position properties of the player's sprite.

The logic for movement changed a bit. The cases are identical except for the direction component. For example, moving left is (-1, 0) and moving right is (1, 0). In the MoveLeft method, I create a Vector2 that describes the desired movement. I set the inMotion field to true saying that we are trying to move. I set the current animation of the sprite. Next, I create a rectangle that describes when we want to stop moving. Following that I use a little LINQ and pattern matching to grab the collision layer. Now, I grab the collisions/ The next step is to check if the collision contains a key that matches the tile that we are trying to move into. If it does not, I set the Player.Tile to the direction we want to move into.

The other three cases flow the same way, so I'm not going to over them. In the Update method, I moved the call to the Update method of the encounter just below the call to the Update method of the control manager. After calling the Update method of the encounter's map, I check to see if the encounter is alive. If it is not, I reset the properties of the player's sprite and pop the state off of the stack to return back to the gameplay state. I now update the _timer field by the amount of time that has passed since the last call to the Update method.

If it is the player's turn and the timer is greater than a quarter of a second, I check to see if the mode is Attack. If it is, I call the HandleAttack method. I will get to it shortly. I then exit the method. I check to see if the mode is Movement. If it is, I check to see if the F key has been released. If it is, I set the mode to Attack and enqueue a message asking where the player wants to attack. So, why a queue? It makes sense because we want the first item to disappear once there are more than four items in the queue. True, we could call the remove method of a list, but a queue fits. After queuing the message, I exit the method. In the if statements where I check for movement, I queue a message that the player is moving in that direction. I call the appropriate Move method, set the turn to the enemy and reset the timer.

If it is not the player's turn and the timer is greater than a quarter of a second, I set the IsAnimating property of the player's sprite to false and call a method HandleEnemies, which I will get to shortly. If the player has reached the end of their turn, in addition to updating the position, I set IsAnimating to false. If the player collides with an object, I add the message Ouch! to the queue.

In the HandleAttack method I check to see if the _timer field is greater than half a second. If it is, I check to see if any of the direction keys have been pressed. If they have, I reset the timer to zero, pass the turn to the enemy side, queue the message that they are attacking in that direction, call a method DoAttack that will do the actual attack, and reset the _mode field to Movement.

The DoAttack method calculates which tile the player is attacking. It then creates a rectangle that describes the tile. I then loop over the enemies on the map. If the enemy's position and the attack rectangle collide, I enqueue a message that the attack was successful. I then decrement the enemy's

health and set the enemy's new health to be the new value. If the attack has missed, I display a message. Currently, HandleEnemies just resets the turn.

In the Draw method, I calculate where to draw the message texture. I then loop over all of the messages and draw them. In the Show method, I reset the turn and timer fields.

The Encounter class also changed. Replace that class with this new version.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Rpglibrary.TileEngine;
using RpgLibrary.Characters;
using RpgLibrary.TileEngine;
using SharpFont;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RpgLibrary
{
    public class Encounter
    {
        public List<ICharacter> Allies { get; set; } = new();
        public List<ICharacter> Enemies { get; set; } = new();
        public TileMap Map { get; set; }

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

        public Encounter(ICharacter player)
        {
            Allies.Add(player);
        }

        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);
            }
        }

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera)
        {
            Map?.Draw(gameTime, spriteBatch, camera);

            foreach (var character in Enemies)
            {
```

```
                character.Draw(spriteBatch);
            }

            foreach (var character in Allies)
            {
                character.Draw(spriteBatch);
            }
        }

        public void RandomMap(ContentManager content)
        {
#if DEBUG
            Random random = new(113399);
#else
            Random random = new();
#endif
            Texture2D texture = content.Load<Texture2D>(@"Tiles/tileset1");

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

            TileLayer layer = new(1280 / Engine.TileWidth + 1, 720 / Engine.TileHeight +
1);

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

            layer = new(1280 / Engine.TileWidth + 1, 720 / Engine.TileHeight + 1);
            CollisionLayer collisions = new();

            for (int y = 0; y < 720 / Engine.TileHeight + 1; y++)
                for (int x = 0; x < 1280 / Engine.TileWidth + 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 / Engine.TileWidth);
                    y = random.Next(1 + 720 / Engine.TileHeight);
                } 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.ContainsKey(new Rectangle(new(x * En-
gine.TileWidth, y * Engine.TileHeight), new(Engine.TileWidth, Engine.TileHeight))));

                if (!collisions.Collisions.ContainsKey(new Rectangle(new(x * Engine.Tile-
Width, y * Engine.TileHeight), new(Engine.TileWidth, Engine.TileHeight))))
                {
                    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);
        }
    }
}
```

The changes were all in the RandomMap method. The first thing I did was wrap the creation of the Random object in a preprocessor check to see if we are running DEBUG or RELEASE. If we are running DEBUG, I create the random variable using the value 113399. Why? While we want a random map, we also want it consistent for debugging. This way, we can easily tell what changed. Why 113399? Well, 11 times 3 is 33 times 3 is 99. Instead of using 64 everywhere, I use the TileWidth and TileHeight properties. When checking to see if the splatter tile is under the player or enemy, I also check to see that there is not a collision at that tile. For some reason, that sometimes fails. So, there is an if to check if a collision exists before adding the collision to the dictionary. This should never happen, in theory. In practice, it would sometimes happen.

If you build and run now, you can move around the map, attack and kill the mob. The mob is still pretty but ultimately useless. Also, there are no rolls to see if you hit, the bat dodges, rolls for damage, etc. We can start with rolls for successful attacks and dodges. Also, I will make the bat attack back, but not move, yet. So, I will suggest you place several points in agility when you create your character.

First, I want to add a method to the Helper class. I moved the RollD10 method from the Mob class. Replace the Helper class with this new version.

```
using Microsoft.Xna.Framework;
using RpgLibrary.Characters;
using System;
using System.Collections.Generic;
using System.Reflection;
using System.Text;

namespace SharedProject
{
    public static class Helper
    {
        private static readonly Random _random = new();

        public static bool RollDie(ICharacter character, string attribute)
        {
            PropertyInfo info = character.GetType().GetProperty(attribute);

            if (info != null)
            {
                if (info.PropertyType == typeof(Int32))
                {
                    int roll = Helper.Random.Next(1, 11);

                    if (roll == 10)
                    {
                        return false;
                    }
```

```
                    if (roll == 1)
                    {
                        return true;
                    }

                    if (int.TryParse(info.GetValue(character).ToString(), out int value))
                    {
                        return (roll <= value);
                    }
                }
            }

            return false;
        }

        public static Vector2 NearestInt(Vector2 vector2)
        {
            return new((int)vector2.X, (int)vector2.Y);
        }

        public static Point V2P(Vector2 vector2)
        {
            return new((int)vector2.X, (int)vector2.Y);
        }

        public static Random Random
        {
            get { return _random; }
        }
    }
}
```

So, a couple of using statements to bring classes into scope. There were a few changes to the method. It takes as a parameter an ICharacter and a string for the attribute to be rolled against. It flows pretty much the same as before. The differences are how I get the property and how I check to see what type the attribute is. I use the PropertyType property and compare it to Int32.

Back in the EncounterState, let's implement rolling for attack and dodge in the DoAttack method. Replace the method with this new version.

```
private void DoAttack(Point direction)
{
    Point target = Player.Tile + direction;
    Rectangle destination = new(target, new(Engine.TileWidth, Engine.TileHeight));

    for (int i =  0; i < encounter.Enemies.Count; i++)
    {
        var enemy = encounter.Enemies[i];

        Point enemyTile = new((int)((Mob)enemy).AnimatedSprite.Position.X / Engine.Tile-
Width,
            (int)((Mob)enemy).AnimatedSprite.Position.Y / Engine.TileHeight);

        Rectangle enemyDestination = new(enemyTile, new(Engine.TileWidth, Engine.Tile-
Height));
```

```
        if (enemyDestination.Intersects(destination) && Helper.RollDie(((ICharac-
ter)Player),"Agility"))
        {
            if (!Helper.RollDie(enemy, "Agility"))
            {
                _messages.Enqueue("   Enemy was hit...");

                AttributePair health = new(enemy.Health.Current);

                health.Current -= Helper.Random.Next(1, 7);
                enemy.Health = health;
            }
            else
            {
                _messages.Enqueue("   Enemy dodges your attack...");
            }
        }
        else
        {
            _messages.Enqueue("   Your attack fell upon empty air...");
        }
    }
}
```

First, I calculate which tile the player is attacking. Then, I loop over the enemies on the map. I set a local variable to the current enemy in the loop. I then create a Point that represents the tile the enemy is in. Following the point, I create a rectangle that describes the enemy's location. If the player's attack tile intersects with the enemy's tile, I call the RollDie method to check if the attack is successful. If the attack was successful, I check to see the bat dodges the attack by rolling against its agility score. If the attack was successful, I display a message and decrement the enemy's health. If the enemy dodges, I display a message saying that. If the attack misses, I display that as a message.

The last thing I'm going to cover is the enemy attacking back. I'm going to park the enemy movement in the next tutorial. Replace the HandleEnemies method with this new version.

```
private void HandleEnemies()
{
    _turn = !_turn;
    _timer = 0;

    foreach (ICharacter c in encounter.Enemies)
    {
        Mob mob = c as Mob;
        Point distance = mob.AnimatedSprite.Tile - Player.Tile;

        if ((Math.Abs(distance.X) == 1 && Math.Abs(distance.Y) == 0) ||
            (Math.Abs(distance.Y) == 1 && Math.Abs(distance.X) == 0))
        {
            bool roll = Helper.RollDie(c, "Agility");

            if (roll)
            {
                if (!Helper.RollDie(Player, "Agility"))
                {
                    _messages.Enqueue($"The {c.Name} swings and hits...");
                }
```

```
                else
                {
                    _messages.Enqueue($"You nimbly dodge the {c.Name}'s attack");
                }
            }
            else
            {
                _messages.Enqueue($"The {c.Name} swings and misses");
            }
        }
    }
}
```

There is now a loop that will loop over all of the enemies in the encounter. The enemy is cast to a Mob object. I get a point that is the difference between the player and the mob's tile. I then check to see if the distance in the X coordinate is one and that Y is zero, or the distance of the Y coordinate is one and that the X is zero. This means that the enemy is one tile away horizontally and in the same column, or they are one tile away vertically and in the same row. If that is true, I call the Roll method of the Helper class, passing in the loop variable and Agility. If it hits, I roll the player's agility. If that fails, the enemy has hit successfully, and I queue a message. What I should do is queue a message and inflict damage. If the player dodged, I queue a message saying that. Finally, I queue a message that the attack has failed.

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