Eyes of the Dragon Tutorials
Part 56
Going Real-Time Part Two

I'm writing these tutorials for the MonoGame 3.8 framework using Visual Studio 2019. 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 page of my web blog. I will be making each version of the project available on GitHub here. It will be included on the page that links to the tutorials.

In this tutorial I will be continuing with real-time combat. In the last tutorial I wrote about the player attacking in real-time, and mobs chasing the player if the player gets too close. What I didn't cover is mobs attacking the player. That is a little trickier than the player attacking mobs. If it is not done correctly, the code will get messy very fast. I will be placing the majority of the code in the Mob class, to separate concerns. So, let's get started.

The first thing to do is to add some fields to the Mob class, common to all mobs. Add the following fields to the Mob class.

```
protected double attackTimer;
protected bool isAttacking;
protected int attackingDirection;
```

The fields should be familiar from the last tutorial. The first, attackTimer, is the amount of time that has elapsed since the last attack. The following field, isAttacking, tells if the mob is currently attacking. Finally, is the attackingDirection field. That field has what direction the mob is attacking. It will range from 0 to 3 where 0 is up, 1 is right, 2 is down, and 3 is left. Why didn't I include Texture2D fields for the weapons? I'm glad that you asked. If I had there would have been a lot of duplication in the child classes. If I placed a Texture2D for the four directions and I had one hundred mobs I'd have four hundred textures. Well, why didn't you make them static then so they are shared by all instances? The reason for that is if I want different weapons for different types of mobs they'd be overridden each time I load a different mob. The solution is to have each individual mob have four static fields. This way there is a reduction in the number of textures needed, and different mobs can have different weapons.

While the Mob class is open, we need to adjust the constructor and add two new abstract methods. Change the constructor to the following and add these abstract methods.

```
public Mob(Entity entity, AnimatedSprite sprite, Game game)
{
    this.entity = entity;
    this.sprite = sprite;
}

public abstract void ShouldAttack(AnimatedSprite sprite);
public abstract void DoAttack(AnimatedSprite sprite, Entity entity);
```

In the constructor I just included a Game parameter because I will require one in the child classes to

load content. The first abstract method, ShouldAttack, will be called to determine if the mob should attack the player. The second, DoAttack, is called to determine if there is a successful attack and apply the damage to an entity. I went that route because eventually I will allow the player to have pets and companions. It is better to include that functionality now rather than implement it at a further date.

Now, in the Bandit class we will implement the attacking behaviour. This class has really changed, so I will give you the code for the entire class. Replace the existing Bandit class with the following code.

```
using MGRpgLibrary.SpriteClasses;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RpgLibrary;
using RpgLibrary.CharacterClasses;
using RpgLibrary.ItemClasses;
using System;
using System.Collections.Generic;
using System.Text;

namespace MGRpgLibrary.Mobs
{
    public class Bandit : Mob
    {
        static Texture2D swordUp;
        static Texture2D swordRight;
        static Texture2D swordDown;
        static Texture2D swordLeft;
        static Game _game;
        Rectangle sword;

        public Bandit(Entity entity, AnimatedSprite sprite, Game game)
            : base(entity, sprite, game)
        {
            _minGold = 20;
            _maxGold = 50;
            _xpValue = 100;

            if (_game == null)
            {
                _game = game;
                LoadContent();
            }
        }

        private void LoadContent()
        {
            swordDown = _game.Content.Load<Texture2D>("ObjectSprites/sword-down");
            swordLeft = _game.Content.Load<Texture2D>("ObjectSprites/sword-left");
            swordRight = _game.Content.Load<Texture2D>("ObjectSprites/sword-right");
            swordUp = _game.Content.Load<Texture2D>("ObjectSprites/sword-up");
        }

        public override void Update(GameTime gameTime)
        {
            attackTimer += gameTime.ElapsedGameTime.TotalSeconds;

            if (attackTimer >= 0.25)
            {
                isAttacking = false;
```

```
    }

    base.Update(gameTime);
}

public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    if (isAttacking)
    {
        switch (attackingDirection)
        {
            case 0:
                sword = new Rectangle(
                    (int)Sprite.Position.X + (32 - swordUp.Width) / 2,
                    (int)Sprite.Position.Y - swordUp.Height,
                    swordUp.Width,
                    swordUp.Height);
                spriteBatch.Draw(swordUp, sword, Color.White);
                break;
            case 2:
                sword = new Rectangle(
                    (int)Sprite.Position.X + (32 - swordDown.Width) / 2,
                    (int)Sprite.Position.Y + 32,
                    swordDown.Width,
                    swordDown.Height);
                spriteBatch.Draw(swordDown, sword, Color.White);
                break;
            case 1:
                sword = new Rectangle(
                    (int)Sprite.Position.X + 32,
                    (int)Sprite.Position.Y + (32 - swordRight.Height) / 2,
                    swordRight.Width,
                    swordRight.Height);
                spriteBatch.Draw(swordRight, sword, Color.White);
                break;
            case 3:
                sword = new Rectangle(
                    (int)Sprite.Position.X - swordLeft.Width,
                    (int)Sprite.Position.Y + (32 - swordLeft.Height) / 2,
                    swordLeft.Width,
                    swordLeft.Height);
                spriteBatch.Draw(swordLeft, sword, Color.White);
                break;
        }
    }

    base.Draw(gameTime, spriteBatch);
}

public override void Attack(Entity source)
{
    if (Mechanics.RollDie(DieType.D20) >= 10 + Mechanics.GetModifier(entity.Dexterity))
    {
        entity.ApplyDamage(source.MainHand);
    }
}

public override void DoAttack(Entity target)
{
    if (Mechanics.RollDie(DieType.D20) >= 10 + Mechanics.GetModifier(target.Dexterity))
```

```
    {
        target.ApplyDamage(entity.MainHand);
    }
}

public override void ShouldAttack(AnimatedSprite sprite)
{
    float distance = Vector2.Distance(sprite.Center, Sprite.Center);

    if (distance < sprite.Width * 2 && attackTimer >= 1.25 && !isAttacking)
    {
        isAttacking = true;
        attackTimer = 0;

        if (Sprite.Position.X < sprite.Position.X)
        {
            attackingDirection = 1;
        }
        else if (Sprite.Position.X > sprite.Position.X)
        {
            attackingDirection = 3;
        }
        else if (Sprite.Position.Y < sprite.Position.Y)
        {
            attackingDirection = 2;
        }
        else
        {
            attackingDirection = 0;
        }

        switch (attackingDirection)
        {
            case 0:
                sword = new Rectangle(
                    (int)Sprite.Position.X + (32 - swordUp.Width) / 2,
                    (int)Sprite.Position.Y - swordUp.Height,
                    swordUp.Width,
                    swordUp.Height);
                break;
            case 2:
                sword = new Rectangle(
                    (int)Sprite.Position.X + (32 - swordDown.Width) / 2,
                    (int)Sprite.Position.Y + 32,
                    swordDown.Width,
                    swordDown.Height);
                break;
            case 1:
                sword = new Rectangle(
                    (int)Sprite.Position.X + 32,
                    (int)Sprite.Position.Y + (32 - swordRight.Height) / 2,
                    swordRight.Width,
                    swordRight.Height);
                break;
            case 3:
                sword = new Rectangle(
                    (int)Sprite.Position.X - swordLeft.Width,
                    (int)Sprite.Position.Y + (32 - swordLeft.Height) / 2,
                    swordLeft.Width,
                    swordLeft.Height);
```

```
                    break;
                }
            }
        }

        public override void DoAttack(AnimatedSprite sprite, Entity entity)
        {
            if (sword.Intersects(sprite.Bounds) && isAttacking)
            {
                if (Entity.MainHand != null && Entity.MainHand.Item is Weapon)
                {
                    entity.ApplyDamage(Entity.MainHand);
                }

                isAttacking = false;
            }
        }
    }
}
```

This is a much more complex class. First, I added some using statements to bring classes into scope. Most notable ones are for the MonoGame framework and MonoGame framework graphics classes. As I mentioned above, I included the static fields for the bandit's sword textures. I also included a static field for a Game reference. There is also a Rectangle field that describes the bandit's sword, if it is attacking. I suppose I could have added this to the base class as it will be common to all sub-classes.

The constructor now takes the required Game parameter and calls the constructor of the base class passing it the parameter. If the _game field is null I set it to the value passed in and call the LoadContent method. In the LoadContent method I load the textures for the swords.

In the Update method I increment the attackTime field by the total seconds since the last game update. If it is greater than a quarter of a second I set isAttacking to false. This way the bandit is only actually attacking for a quarter of a second like the player. However, as you saw, it can only attack ever 1.25 seconds. That gives the player the opportunity to get in two or three hits to every one by the bandit.

In the Draw method I check to see if isAttacking is true. If it is, we need to draw the sword. Like I did for the player there is a switch on the attacking direction. It flows the same way as it did for the player, and the cases are similar so I am only going to go over the first one. I create a rectangle centering the sword horizontally over the sprite. I then position the sword above the sprite by taking the Y property of the sprite's coordinates and subtracting the height of the sword. I then draw the sword.

The Attack method is from the old CombatScreen and is unchanged, and will be unused. The DoAttack method is also an implementation of old abstract method and is unused now that we are doing real-time combat.

The ShouldAttack method is new and takes an AnimatedSprite object as an argument. It calculates the distance between the center of the bandit's sprite and the sprite passed in. If that distance is less than two times the width of the target, that attackTimer is greater than 1.25, and isAttacking is false I take action. I set isAttacking to true and reset the attackTimer to zero. If the X property of the bandit's

sprite's position is less than the X property of the sprite's position, I set the attackingDirection to 1, or right. If X property is not less than but is greater than X property of the sprite's position I set the attacking direction to 3, or left. Similarly, if the Y property of the bandit's sprite's position is less than Y property sprite's position I set attackingDIrection to 2, or down. If none of the other cases were true we attempt to attack up.

This is really going to bias attacking left and right over up and down. If you are to the left of the attacking bandit at all they will attack left, even if you are closer to them above. This isn't meant to be a perfect solution at the moment. More, it is a demonstration on how to trigger an attack if certain conditions are met. I will perfect an algorithm for determining an attack and present it in a future tutorial.

The overload of the DoAttack method that takes an AnimatedSprite and Entity for arguments checks to see if the sword rectangle intersects with the bounds of the sprite passed in and that the isAttacking field is true. If both are true it checks to see if there is something in the main hand of the bandit, and that it is a Weapon. If that is true I call the Apply damage method of the entity passed in. I also set isAttacking to false.

Changing the constructor of the Mob and Bandit classes I broke the CharacterGeneratorScreen. In particular the LoadWorld method. Replace the LoadWorld method of the CharacterGeneratorScreen with the following version.

```
private void LoadWorld()
{
    RpgLibrary.WorldClasses.LevelData levelData =
        Game.Content.Load<RpgLibrary.WorldClasses.LevelData>(@"Game\Levels\Starting Level");

    RpgLibrary.WorldClasses.MapData mapData =
        Game.Content.Load<RpgLibrary.WorldClasses.MapData>(@"Game\Levels\Maps\" +
levelData.MapName);

    CharacterLayerData charData =
        Game.Content.Load<CharacterLayerData>(@"Game\Levels\Chars\Starting Level");
    CharacterLayer characterLayer = new CharacterLayer();
    MobLayer mobLayer = new MobLayer();

    TileMap map = TileMap.FromMapData(mapData, Game.Content);

    foreach (var c in charData.Characters)
    {
        Character character;

        if (c.Value is NonPlayerCharacterData data)
        {
            Entity entity = new Entity(c.Value.Name, c.Value.EntityData, c.Value.Gender,
EntityType.NPC);

            using (Stream stream = new FileStream(c.Value.TextureName, FileMode.Open,
FileAccess.Read))
            {
                Texture2D texture = Texture2D.FromStream(GraphicsDevice, stream);
                AnimatedSprite sprite = new AnimatedSprite(texture,
AnimationManager.Instance.Animations)
```

```
                {
                    Position = new Vector2(c.Key.X * Engine.TileWidth, c.Key.Y *
Engine.TileHeight)
                };

                character = new NonPlayerCharacter(entity, sprite);

                ((NonPlayerCharacter)character).SetConversation(
                    data.CurrentConversation);
            }

            characterLayer.Characters.Add(c.Key, character);
        }
    }

    map.AddLayer(characterLayer);
    map.AddLayer(mobLayer);

    Level level = new Level(map);

    ChestData chestData = Game.Content.Load<ChestData>(@"Game\Chests\Plain Chest");

    Chest chest = new Chest(chestData);

    BaseSprite chestSprite = new BaseSprite(
        containers,
        new Rectangle(0, 0, 32, 32),
        new Point(10, 10));

    ItemSprite itemSprite = new ItemSprite(
        chest,
        chestSprite);

    level.Chests.Add(itemSprite);

    World world = new World(GameRef, GameRef.ScreenRectangle);

    world.Levels.Add(level);
    world.CurrentLevel = 0;

    AnimatedSprite s = new AnimatedSprite(
        GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
        AnimationManager.Instance.Animations)
    {
        Position = new Vector2(0 * Engine.TileWidth, 5 * Engine.TileHeight)
    };

    EntityData ed = new EntityData("Eliza", 1, 10, 10, 10, 10, 10, 10, "20|CON|12", "16|WIL|
16",
        "0|0|0");

    Entity e = new Entity("Eliza", ed, EntityGender.Female, EntityType.NPC);

    NonPlayerCharacter npc = new NonPlayerCharacter(e, s);

    npc.SetConversation("eliza1");
    //world.Levels[world.CurrentLevel].Characters.Add(npc);

    s = new AnimatedSprite(
        GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
```

```
        AnimationManager.Instance.Animations)
    {
        Position = new Vector2(10 * Engine.TileWidth, 0)
    };

    ed = new EntityData("Barbra", 2, 10, 10, 10, 10, 10, 10, "20|CON|12", "16|WIL|16", "0|0|
0");

    e = new Entity("Barbra", ed, EntityGender.Female, EntityType.Merchant);

    Merchant m = new Merchant(e, s);
    Texture2D items = Game.Content.Load<Texture2D>("ObjectSprites/roguelikeitems");
    m.Backpack.AddItem(GameItemManager.GetItem("Long Sword"));
    m.Backpack.AddItem(GameItemManager.GetItem("Short Sword"));
    m.Backpack.AddItem(GameItemManager.GetItem("Apprentice Staff"));
    m.Backpack.AddItem(GameItemManager.GetItem("Acolyte Staff"));
    m.Backpack.AddItem(GameItemManager.GetItem("Leather Armor"));
    m.Backpack.AddItem(GameItemManager.GetItem("Chain Mail"));
    m.Backpack.AddItem(GameItemManager.GetItem("Studded Leather Armor"));
    m.Backpack.AddItem(GameItemManager.GetItem("Light Robes"));
    m.Backpack.AddItem(GameItemManager.GetItem("Medium Robes"));
    world.Levels[world.CurrentLevel].Characters.Add(m);
    ((CharacterLayer)world.Levels[world.CurrentLevel].Map.Layers.Find(x => x is
CharacterLayer)).Characters.Add(new Point(10, 0), m);
    GamePlayScreen.World = world;

    for (int i = 0; i < 25; i++)
    {
        ed = new EntityData("Bandit", 1, 10, 12, 12, 10, 10, 10, "20|CON|10", "12|WIL|12", "0|
0|0");

        e = new Entity("Bandit", ed, EntityGender.Male, EntityType.Monster);

        s = new AnimatedSprite(
            GameRef.Content.Load<Texture2D>(@"PlayerSprites/malerogue"),
            AnimationManager.Instance.Animations);

        Mob mob = new Bandit(e, s, GameRef);

        Rectangle r = new Rectangle(Mechanics.Random.Next(10, 50) * 32,
Mechanics.Random.Next(10, 50) * 32, 32, 32);

        mob.Sprite.Position = new Vector2(r.X, r.Y);

        if (!mobLayer.Mobs.ContainsKey(r))
        {
            mobLayer.Mobs.Add(r, mob);
        }

        mob.Entity.Equip(GameItemManager.GetItem("Short Sword"));
        mob.Drops.Add(GameItemManager.GetItem("Short Sword"));
        mob.Drops.Add(GameItemManager.GetItem("Minor Healing Potion"));
    }
}
```

Before the bandit's will attempt to attack you, they need to be told to attempt to attack you. That will be done in the Update method of the GamePlayScreen class. Also, I added the code for the case that the player is killed by a bandit and push the game over state onto the stack. Replace that method with the following code.

```
public override void Update(GameTime gameTime)
{
    world.Update(gameTime);
    player.Update(gameTime);
    player.Camera.LockToSprite(player.Sprite);

    if (InputHandler.KeyReleased(Keys.Space) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One))
    {
        foreach (ILayer layer in World.Levels[World.CurrentLevel].Map.Layers)
        {
            if (layer is CharacterLayer)
            {
                foreach (Character c in ((CharacterLayer)layer).Characters.Values)
                {
                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        c.Sprite.Center);

                    if (distance < Character.SpeakingRadius && c is NonPlayerCharacter)
                    {
                        NonPlayerCharacter npc = (NonPlayerCharacter)c;

                        if (npc.HasConversation)
                        {
                            StateManager.PushState(GameRef.ConversationScreen);

                            GameRef.ConversationScreen.SetConversation(
                                player,
                                npc,
                                npc.CurrentConversation);

                            GameRef.ConversationScreen.StartConversation();
                        }
                    }
                    else if (distance < Character.SpeakingRadius && c is Merchant)
                    {
                        StateManager.PushState(GameRef.ShopScreen);
                        GameRef.ShopScreen.SetMerchant(c as Merchant);
                    }
                }
            }
        }
    }

    MobLayer mobLayer = World.Levels[World.CurrentLevel].Map.Layers.Find(x => x is MobLayer) as
MobLayer;

    if (playerAttacking)
    {
        foreach (var mob in mobLayer.Mobs.Values)
        {
            if (playerSword.Intersects(mob.Sprite.Bounds))
            {
                if (player.Character.Entity.MainHand != null &&
                    player.Character.Entity.MainHand.Item is Weapon)
                {
                    mob.Entity.ApplyDamage(player.Character.Entity.MainHand);
                    playerAttacking = false;
```

```
                    if (mob.Entity.Health.CurrentValue <= 0)
                    {
                        StateManager.PushState(GameRef.LootScreen);
                        GamePlayScreen.Player.Character.Entity.AddExperience(mob.XPValue);
                        GameRef.LootScreen.Gold = mob.GoldDrop;

                        foreach (var i in mob.Drops)
                        {
                            GameRef.LootScreen.Items.Add(i);
                        }
                    }
                }
            }
        }
    }

    foreach (var mob in mobLayer.Mobs.Where(kv => kv.Value.Entity.Health.CurrentValue <=
0).ToList())
    {
        mobLayer.Mobs.Remove(mob.Key);
    }

    foreach (var mob in mobLayer.Mobs.Values)
    {
        mob.DoAttack(player.Sprite, player.Character.Entity);
        mob.ShouldAttack(player.Sprite);
    }

    foreach (var mob in mobLayer.Mobs.Values)
    {
        float distance = Vector2.Distance(player.Sprite.Center, mob.Sprite.Center);

        if (distance < mob.Sprite.Width * 4)
        {
            Vector2 motion = Vector2.Zero;

            if (mob.Sprite.Position.X < player.Sprite.Position.X)
            {
                motion.X = 1;
                mob.Sprite.CurrentAnimation = AnimationKey.Right;
            }

            if (mob.Sprite.Position.X > player.Sprite.Position.X)
            {
                motion.X = -1;
                mob.Sprite.CurrentAnimation = AnimationKey.Left;
            }

            if (mob.Sprite.Position.Y < player.Sprite.Position.Y)
            {
                motion.Y = 1;
                mob.Sprite.CurrentAnimation = AnimationKey.Down;
            }

            if (mob.Sprite.Position.Y > player.Sprite.Position.Y)
            {
                motion.Y = -1;
                mob.Sprite.CurrentAnimation = AnimationKey.Up;
            }
```

```
            if (motion != Vector2.Zero)
            {
                motion.Normalize();
            }

            float speed = 200f;

            motion *= speed * (float)gameTime.ElapsedGameTime.TotalSeconds;

            mob.Sprite.Position += motion;
            mob.Sprite.IsAnimating = true;

            if (mob.Sprite.Bounds.Intersects(player.Sprite.Bounds))
            {
                mob.Sprite.Position -= motion;
            }
        }
        else
        {
            mob.Sprite.IsAnimating = false;
        }
    }
    if (InputHandler.KeyReleased(Keys.I))
    {
        StateManager.PushState(GameRef.InventoryScreen);
    }

    if (InputHandler.KeyReleased(Keys.C))
    {
        StateManager.PushState(GameRef.StatsScreen);
        Visible = true;
    }

    if (Player.Character.Entity.Level < Mechanics.Experiences.Length)
    {
        if (Player.Character.Entity.Experience >=
Mechanics.Experiences[Player.Character.Entity.Level])
        {
            Player.Character.Entity.LevelUp();
            StateManager.PushState(GameRef.LevelScreen);
            Visible = true;
        }
    }

    if (InputHandler.CheckMousePress(MouseButton.Left) && playerTimer > 0.25 && !
playerAttacking)
    {
        playerAttacking = true;
        playerTimer = 0;

        if (player.Sprite.CurrentAnimation == AnimationKey.Up)
        {
            attackDirection = 0;
        }
        else if (player.Sprite.CurrentAnimation == AnimationKey.Right)
        {
            attackDirection = 1;
        }
        else if (player.Sprite.CurrentAnimation == AnimationKey.Down)
        {
```

```
            attackDirection = 2;
        }
        else
        {
            attackDirection = 3;
        }
    }

    if (playerTimer >= 0.25)
    {
        playerAttacking = false;
    }

    playerTimer += gameTime.ElapsedGameTime.TotalSeconds;

    if (player.Character.Entity.Health.CurrentValue <= 0)
    {
        StateManager.PushState(GameRef.GameOverScreen);
    }

    base.Update(gameTime);
}
```

If you build and run now the bandits will chase and attack the player if the player gets close enough. Hopefully, you are able to still kill the bandits. You should because your attack rate is much faster than the bandit's attack rate. Also, their attack radius is wider than yours so they should attack when you are not in range. Between the two you should be able to slay some bandits.

There is still the problem where if you don't get potions you will die pretty quickly. I will remedy that by slowly increasing the player's health over time. What I did was every second call the Heal method of the player's Entity. So, add the following field to the list of fields of the GamePlayState and replace the Update method with the following code.

```
double _healthTimer;

public override void Update(GameTime gameTime)
{
    world.Update(gameTime);
    player.Update(gameTime);
    player.Camera.LockToSprite(player.Sprite);

    if (InputHandler.KeyReleased(Keys.Space) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One))
    {
        foreach (ILayer layer in World.Levels[World.CurrentLevel].Map.Layers)
        {
            if (layer is CharacterLayer)
            {
                foreach (Character c in ((CharacterLayer)layer).Characters.Values)
                {
                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        c.Sprite.Center);

                    if (distance < Character.SpeakingRadius && c is NonPlayerCharacter)
                    {
                        NonPlayerCharacter npc = (NonPlayerCharacter)c;
```

```
                        if (npc.HasConversation)
                        {
                            StateManager.PushState(GameRef.ConversationScreen);

                            GameRef.ConversationScreen.SetConversation(
                                player,
                                npc,
                                npc.CurrentConversation);

                            GameRef.ConversationScreen.StartConversation();
                        }
                    }
                    else if (distance < Character.SpeakingRadius && c is Merchant)
                    {
                        StateManager.PushState(GameRef.ShopScreen);
                        GameRef.ShopScreen.SetMerchant(c as Merchant);
                    }
                }
            }
        }
    }

    MobLayer mobLayer = World.Levels[World.CurrentLevel].Map.Layers.Find(x => x is MobLayer) as
MobLayer;

    if (playerAttacking)
    {
        foreach (var mob in mobLayer.Mobs.Values)
        {
            if (playerSword.Intersects(mob.Sprite.Bounds))
            {
                if (player.Character.Entity.MainHand != null &&
                    player.Character.Entity.MainHand.Item is Weapon)
                {
                    mob.Entity.ApplyDamage(player.Character.Entity.MainHand);
                    playerAttacking = false;

                    if (mob.Entity.Health.CurrentValue <= 0)
                    {
                        StateManager.PushState(GameRef.LootScreen);
                        GamePlayScreen.Player.Character.Entity.AddExperience(mob.XPValue);
                        GameRef.LootScreen.Gold = mob.GoldDrop;

                        foreach (var i in mob.Drops)
                        {
                            GameRef.LootScreen.Items.Add(i);
                        }
                    }
                }
            }
        }
    }

    foreach (var mob in mobLayer.Mobs.Where(kv => kv.Value.Entity.Health.CurrentValue <=
0).ToList())
    {
        mobLayer.Mobs.Remove(mob.Key);
    }
```

```
foreach (var mob in mobLayer.Mobs.Values)
{
    mob.DoAttack(player.Sprite, player.Character.Entity);
    mob.ShouldAttack(player.Sprite);
}

foreach (var mob in mobLayer.Mobs.Values)
{
    float distance = Vector2.Distance(player.Sprite.Center, mob.Sprite.Center);

    if (distance < mob.Sprite.Width * 4)
    {
        Vector2 motion = Vector2.Zero;

        if (mob.Sprite.Position.X < player.Sprite.Position.X)
        {
            motion.X = 1;
            mob.Sprite.CurrentAnimation = AnimationKey.Right;
        }

        if (mob.Sprite.Position.X > player.Sprite.Position.X)
        {
            motion.X = -1;
            mob.Sprite.CurrentAnimation = AnimationKey.Left;
        }

        if (mob.Sprite.Position.Y < player.Sprite.Position.Y)
        {
            motion.Y = 1;
            mob.Sprite.CurrentAnimation = AnimationKey.Down;
        }

        if (mob.Sprite.Position.Y > player.Sprite.Position.Y)
        {
            motion.Y = -1;
            mob.Sprite.CurrentAnimation = AnimationKey.Up;
        }

        if (motion != Vector2.Zero)
        {
            motion.Normalize();
        }

        float speed = 200f;

        motion *= speed * (float)gameTime.ElapsedGameTime.TotalSeconds;

        mob.Sprite.Position += motion;
        mob.Sprite.IsAnimating = true;

        if (mob.Sprite.Bounds.Intersects(player.Sprite.Bounds))
        {
            mob.Sprite.Position -= motion;
        }
    }
    else
    {
        mob.Sprite.IsAnimating = false;
    }
}
```

```
    if (InputHandler.KeyReleased(Keys.I))
    {
        StateManager.PushState(GameRef.InventoryScreen);
    }

    if (InputHandler.KeyReleased(Keys.C))
    {
        StateManager.PushState(GameRef.StatsScreen);
        Visible = true;
    }

    if (Player.Character.Entity.Level < Mechanics.Experiences.Length)
    {
        if (Player.Character.Entity.Experience >=
Mechanics.Experiences[Player.Character.Entity.Level])
        {
            Player.Character.Entity.LevelUp();
            StateManager.PushState(GameRef.LevelScreen);
            Visible = true;
        }
    }

    if (InputHandler.CheckMousePress(MouseButton.Left) && playerTimer > 0.25 && !
playerAttacking)
    {
        playerAttacking = true;
        playerTimer = 0;

        if (player.Sprite.CurrentAnimation == AnimationKey.Up)
        {
            attackDirection = 0;
        }
        else if (player.Sprite.CurrentAnimation == AnimationKey.Right)
        {
            attackDirection = 1;
        }
        else if (player.Sprite.CurrentAnimation == AnimationKey.Down)
        {
            attackDirection = 2;
        }
        else
        {
            attackDirection = 3;
        }
    }

    if (playerTimer >= 0.25)
    {
        playerAttacking = false;
    }

    playerTimer += gameTime.ElapsedGameTime.TotalSeconds;

    if (player.Character.Entity.Health.CurrentValue <= 0)
    {
        StateManager.PushState(GameRef.GameOverScreen);
    }

    if (player.Character.Entity.Health.CurrentValue <
player.Character.Entity.Health.MaximumValue)
```

```
    {
        _healthTimer += gameTime.ElapsedGameTime.TotalSeconds;

        if (_healthTimer > 1)
        {
            _healthTimer = 0;
            player.Character.Entity.Health.Heal(2);
        }
    }

    base.Update(gameTime);
}
```

If you build and run now you will slowly gain health over time, if you are injured. If you kill a bandit you will gain loot. Finally, if you kill enough bandits you will level up.

So, that is going to be it for this tutorial. I accomplished what I intended, and I don't want to venture further in this tutorial. So, please continue to visit my blog, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck with your Game Programming Adventures!

Cynthia