Eyes of the Dragon Tutorials
Part 55
Going Real-Time

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 moving to real-time combat, it has been requested and I like to fill requests that are made. To get started, I want to spread the mobs over a larger area of the map. Otherwise, they will swarm the player when testing combat. To do that, I modified the LoadWorld method of the CharacterGeneratorScreen class. I just increased the range to (10, 10) to (49, 49). Replace the LoadWorld method of the CharacterGeneratorScreen class with the following.

```
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);

        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"));
    }
}
```

That should give a good distribution of mobs for testing. The other change that I made was I set the mob's Position field to the X and Y coordinate of the rectangle.

For the actual combat I am going to cheat a little, because I don't have sprites with attack animations. What I am going to do is use a sword image for all character classes and weapons. Again, that is because I don't have the graphics and I didn't want to go searching online for them. As it is, I did need to do a little bit of a search for some graphics to use. I used this sword at a template: https://opengameart.org/content/medieval-sword. I made it smaller and versions of it pointing in all four cardinal directions. I could have done the diagonals but I will leave that as an exercise. You can download my version from blog at the following address:

https://cynthiamcmahon.ca/blog/downloads/swords.zip. Download the file, extract the swords and add them to the ObjectSprites folder in the content project.

Now that we have the swords in place we need to load them. I did that in the GamePlayScreen class. Add the following fields to the GamePlayScreen and update the LoadContent method to the following.

```
Texture2D swordUp;
Texture2D swordRight;
Texture2D swordDown;
Texture2D swordLeft;

Rectangle playerSword;
bool playerAttacking;
double playerTimer;
int attackDirection;

protected override void LoadContent()
{
    swordUp = GameRef.Content.Load<Texture2D>("ObjectSprites/Sword-Up");
    swordRight = GameRef.Content.Load<Texture2D>("ObjectSprites/Sword-Right");
    swordDown = GameRef.Content.Load<Texture2D>("ObjectSprites/Sword-Down");
    swordLeft = GameRef.Content.Load<Texture2D>("ObjectSprites/Sword-Left");

    base.LoadContent();
}
```

As you can see, I included four other fields. The first, playerSword, is the destination rectangle for the player's sword. The other, playerAttacking, is a bool that says if the player is attacking currently. The next, playerTimer, times between times the player can attack. The last is attackDirection and stores which direction the player attacked in.

The Update method was a major rewrite. I had to remove the code that triggered the combat screen. Also, I needed to add a mechanism to start an attack and finish an attack. Replace the Update method of the GamePlayScreen with the following version.

```
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);
        }

        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;

        base.Update(gameTime);
    }
```

What I did was rip out the old code that dealt with moving to the CombatScreen. In its place, there is a check to see if playerAttacking is true. If it is, I loop over all the mobs in the dictionary of mobs. I then check to see if the playerSword rectangle intersects with the Bound property of the AnimatedSprite class, that I will add shortly. If that is true, I check to see that the player is holding a weapon in their main hand. If they are I call the ApplyDamage method on the mob passing in the weapon. As you can see, there is no long a roll to see if the character hits or not. That will be determined by if the character is close enough to hit. Then, if the mob's health is less than or equal to zero, I push the LootScreen on to the stack like I did in the old CombatScreen. I update the player character's experience next and add the gold. Next I loop over the drops and add them to the loot collection of the LootScreen.

At the end of the method I check to see if the left mouse button has been pressed. I do pressed instead of released because I want the attack to happen at the exact same time as the button is pressed. I also check to see that the timer is greater than 0.25, or a quarter of a second, and that the player currently isn't attacking. If those conditions are all true, I set playerAttacking to true and the playerTimer back to zero. Next is a series of if statements that check the player's current animation. If it is up I set the attackDirection field to 0, 1 if its right, 2 if it is down, and 3 if it is left.

Following finding the attack direction there is a check to see if the amount have time that has passed since the last attack is greater than a quarter of a second. If it is, playerAttacking is set to false. The final change is that playerTimer field is incremented by the total seconds that have elapsed since the last Update call.

Now, to the AnimatedSprite class I added a read-only property that returns a rectangle that describes the sprite's bounds on the screen. Add this property to the AnimatedSprite class.

```
public Rectangle Bounds
{
    get
    {
        return new Rectangle((int)Position.X, (int)Position.Y, Width, Height);
    }
}
```

I think that will all be familiar to you. It takes the position property, cast to integers, along with the Width and Height properties, and returns a new Rectangle.

The next thing that I did was update the Draw method to draw the sword if the player is attacking. Replace the existing Draw method of the GamePlayScreen class with the following code.

```
public override void Draw(GameTime gameTime)
{
    GameRef.SpriteBatch.Begin(
        SpriteSortMode.Deferred,
        BlendState.AlphaBlend,
        SamplerState.PointClamp,
        null,
        null,
        null,
        player.Camera.Transformation);

    base.Draw(gameTime);
    world.DrawLevel(gameTime, GameRef.SpriteBatch, player.Camera);

    if (playerAttacking)
    {
        switch (attackDirection)
        {
            case 0:
                playerSword = new Rectangle(
                    (int)player.Sprite.Position.X + (32 - swordUp.Width) / 2,
                    (int)player.Sprite.Position.Y - swordUp.Height,
                    swordUp.Width,
                    swordUp.Height);
                GameRef.SpriteBatch.Draw(swordUp, playerSword, Color.White);
                break;
            case 2:
                playerSword = new Rectangle(
                    (int)player.Sprite.Position.X + (32 - swordDown.Width) / 2,
                    (int)player.Sprite.Position.Y + 32,
                    swordDown.Width,
                    swordDown.Height);
                GameRef.SpriteBatch.Draw(swordDown, playerSword, Color.White);
                break;
            case 1:
                playerSword = new Rectangle(
                    (int)player.Sprite.Position.X + 32,
                    (int)player.Sprite.Position.Y + (32 - swordRight.Height) / 2,
                    swordRight.Width,
                    swordRight.Height);
                GameRef.SpriteBatch.Draw(swordRight, playerSword, Color.White);
                break;
            case 3:
                playerSword = new Rectangle(
                    (int)player.Sprite.Position.X - swordLeft.Width,
                    (int)player.Sprite.Position.Y + (32 - swordLeft.Height) / 2,
                    swordLeft.Width,
                    swordLeft.Height);
                GameRef.SpriteBatch.Draw(swordLeft, playerSword, Color.White);
                break;
        }
    }
    player.Draw(gameTime, GameRef.SpriteBatch);

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

What the new code does is check to see the playerAttacking field is true. If it is there is a switch on the attackDirection field. If the case is zero, I create a rectangle that is above the player. The X coordinate is the player's X coordinate plus the width of the sprite, minus the width of the up sword divided by two to center the sword horizontally. Its Y coordinate is the player's Y coordinate minus the height of the up sword. For all rectangles the width and height are the width and height of the matching swords. I then draw the up sword.

The down sword is similar to the up sword. It is centered horizontally as well. The difference is that its Y coordinate is set to the Y value of the player plus the height of the sprite.

The right and left cases work the same way as the up and down cases. The difference is that the swords are centered vertically instead of horizontally. The left sword is positioned to the left of the player by taking their X coordinate and subtracting the width of the sword. The right sword is positioned by taking the X coordinate and adding the width of the player.

If you build and run now. You can fight the mobs, kill them and gain experience like before. However, the mobs do not fight back.

The last thing I am going to tackle in this tutorial is having the mobs chase the player. I am going to handle attacking in the next tutorial because I need time to think out how I am going to handle that. To have the mobs chase the player I modified the Update method of the GamePlayScreen class. 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)
    {
        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;

    base.Update(gameTime);
}
```

What the new code does is loop over all of the mobs on the map, after the dead mobs have been removed. I then calculate the distance between the two mobs' centers. If it is less than four times the width of the sprite I create a local Vector2 variable motion. This variable will hold the direction of the sprite to follow the player. If the mob's X coordinate is less then the player's X coordinate I set motion's X property to 1 so the mob will move right and set its CurrentAnimation property to right. If the mob's X coordinate is great than the player's X coordinate we want to move left. Similarly, if the mob's Y coordinate is less than the player's Y coordinate we want to move down. Finally, if the mob's Y coordinate is greater than the player's Y coordinate we want to move up.

Technically motion should never be the zero vector but just in case I test to see if it is the zero vector before normalizing it. I then calculate the distance that the sprite will travel in the selected direction. I then update the sprite's position and set it to be animating. If the mob's bounds intersect with the player's bounds I undo the position update.

If the distance between the two sprite is not less then four times the width I set its IsAnimating property to be false. If you build and run now the mobs will start to chase you if you get too close. They move slower than you so you can out run them.

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