Eyes of the Dragon Tutorials
Part 58
Going Real-Time Part Four

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 particular, I will be performing the actual spell or talent the player has activated. First, I want to change things a bit. I don't want the mouse cursor visible while targeting. I only want the target displayed where the mouse should be. It will be a little less confusing for the player. I will handle that in the Draw method of the GamePlayScreen. Also, I want to change the way the target cursor is drawn. I want to draw it centered on the mouse cursor rather than it being in the upper left corner. Replace that method with the following code.

```
public override void Draw(GameTime gameTime)
{
    GameRef.IsMouseVisible = !targeting;

    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);

    if (targeting)
    {
        Color tint = Color.White;
        float scale = (float)currentActivation.AreaOfEffect / target.Width;

        Vector2 targetPosition = Player.Camera.Position + InputHandler.MouseAsVector2;
        targetPosition -= new Vector2(target.Width, target.Height) * scale / 2;

        if (Vector2.Distance(Player.Sprite.Center, targetPosition) > currentActivation.Range)
        {
            tint = Color.Black;
        }

        GameRef.SpriteBatch.Draw(
            target,
            targetPosition,
            null,
            tint,
            0f,
            new Vector2(),
            scale,
            SpriteEffects.None,
            1f);
    }
    GameRef.SpriteBatch.End();
}
```

All I did was set GameRef.IsMouseVisible to !target. That way, the cursor is not visible when the target is and vice versa. To center the target texture over the mouse coordinates I create a Vector2 that is the height and width of the texture. I multiply that by the scale and divide that by 2 because the radius of the circle is half the width of the texture.

Now, it is going to get a little messy, sorry. The first step will be to update the condition where I check if the left mouse button is clicked while targeting something. I first want to check that the target is within range. That is what is going to be a little messy. Update the Update method to the following code and add the following using statements.

```
using RpgLibrary.SpellClasses;
using RpgLibrary.TalentClasses;
using RpgLibrary.EffectClasses;

public override void Update(GameTime gameTime)
{
    if (targeting)
    {
        List<BaseEffect> effects = currentActivation is Spell
            ? ((Spell)currentActivation).Effects
            : ((Talent)currentActivation).Effects;

        if (effects[0].TargetType == TargetType.Self)
        {
            targeting = false;
            ActivateSelfEffect(effects);
            return;
        }

        if (InputHandler.KeyPressed(Keys.Escape))
        {
            targeting = false;
        }

        if (InputHandler.CheckMousePress(MouseButton.Left))
        {
            castTime = 0;

            if (castTime >= currentActivation.CastTime)
            {
                MobLayer mobLayer = (MobLayer)World.CurrentMap.Layers.Find(x => x is MobLayer);
                Vector2 mouse = InputHandler.MouseAsVector2;
                float scale = (float)currentActivation.AreaOfEffect / target.Width;

                Vector2 targetPosition = Player.Camera.Position + InputHandler.MouseAsVector2;

                foreach (Mob m in mobLayer.Mobs.Values)
                {
                    float distance = Vector2.Distance(targetPosition, m.Sprite.Center);

                    if (distance > currentActivation.AreaOfEffect / 2)
                    {
                        continue;
                    }

                    ActivateEnemyEffect(effects, m);

                    targeting = false;
                }

            }
        }

        return;
    }

    world.Update(gameTime);
    player.Update(gameTime);

    player.Camera.LockToSprite(player.Sprite);
```

```
    HandleHotKeyInput();

    HandleConversation();

    HandleMobs(gameTime);

    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);
            player.Character.Entity.Mana.Heal(2);
            player.Character.Entity.Stamina.Heal(2);
        }
    }

    base.Update(gameTime);
}
```

Messy, but not as messy as I had feared. I have targeting as part of the BaseEffect class where it would have made more sense to include it in the Activation class and have spells and talents inherit from that class. Instead of going through all the hassle to do that, I found a slick way to get the effects without having to have huge if-statements if what is being activated is a spell or a talent. I use the ? operator on if the currentActivation is a Spell. If it is, I get the list of BaseEffects by casting to a Spell and a Talent otherwise. I then check to see if the TargetType of the first effect is TargetType.Self, I'm enforcing that all effects in a spell or talent affect the same target. If it is a self targeting spell I set targeting to false, call a new method I will add shortly, and return.

The method flow as before until it gets to the check for the left button. Like before it sets castTime to 0. Then it checks to see if castTime is greater than or equal to the CastTime property of the current activation. I am probably going to rethink this in a future tutorial. I then get the MobLayer using the same method as before. I grab the position of mouse in case I need it. I then get the targetPosition of the mouse. Now I loop over all of the mobs on the mob layer. I calculate the distance of the spell in relation to the center of the target mob. If the distance is half the area of effect I set targeting to false and call a new method that I will add shortly. I am going to rethink that and redo it in a future tutorial. Something just feels wrong about it. However, I can't put my finger on what.

The two activation methods are similar and simpler than I had originally anticipated. Here is the code for the two new methods.

```
private void ActivateEnemyEffect(List<BaseEffect> effects, Mob m)
{
    foreach (BaseEffect e in effects)
    {
        e.Apply(m.Entity);
    }
}

private void ActivateSelfEffect(List<BaseEffect> effects)
{
```

```
    foreach (BaseEffect e in effects)
    {
        e.Apply(Player.Character.Entity);
    }
}
```

They both just iterate over effects. The one for enemy effects calls the Apply method of the effect passing in the Entity property of the mob. The one for the player calls the Apply method of the effect and passes in the Entity property of the player's character.

If you build and run now you can activate the first hot key. You can dismiss it by pressing the escape key. You can also cast spells now if the target is within range.

That is it for this tutorial. I accomplished what I had planned on, and I don't want to get into something new at this point. I accomplished most of 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