

Eyes of the Dragon Tutorials

Part 32

Collisions

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.

So far we've got the player's sprite wandering around the screen. There is one thing missing though. The player can walk through objects and the player can't interact with objects. In this tutorial I will be covering the basics of sprite to tile collision. To do that we are going to add a special layer called a collision layer. This layer indicates how the player will interact with the tiles. Such as where they can walk through it or not. So, fire up version of Visual Studio and let's get started.

First, right click the **EyesOfTheDragon** project and select **Set as Startup Project** to make sure that when we run the project that it will start rather than one of the editors. Now, in the **MGRpgLibrary** project right click the **TileEngine** folder, select Add and then Class. Name this new class **CollisionLayer**. Here is the base code for the **CollisionLayer** class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;

namespace MGRpgLibrary.TileEngine
{
    public enum CollisionType
    {
        Passable = 0,
        Impassable = 1
    }

    public class CollisionLayer
    {
        public const int CollisionRadius = 4;
        private readonly Dictionary<Point, CollisionType> collisions;

        public Dictionary<Point, CollisionType> Collisions
        {
            get { return collisions; }
        }

        public CollisionLayer()
        {
            collisions = new Dictionary<Point, CollisionType>();
        }
    }
}
```

First thing you will see is that I added an enumeration to the class, **CollisionType**, that has two values

Passable and **Impassable**. While **Passable** won't always be used it was included because it will be helpful in the future. It is possible to have other types of collision associated with a tile. For example, you could have a collision of type Poison that damages the character that enters the tile. Only your imagination is the limit really.

Second, I included a constant **CollisionRadius**. I added this because sometimes you might want a sprite to travel down a path that is one tile wide or enter a door that is one tile wide. Without the collision radius the player would need to line up their sprite to the pixel of the object, which would be very maddening. Using this collision radius value a few pixels will be taken off to give the tiles a bit of padding in terms of collision detection. 4 is fairly good for the tile width and height that we have. If you have smaller tiles you will want to decrease the value and increase it if you have larger tiles. Either 1/8 or 1/16 is typically a good value.

Collisions for the map will be stored in a **Dictionary<Point, CollisionType>**. **Point** is the key and **CollisionType** is the value where **Point** is the tile that the collision will be applied to. There is a read property to expose the collisions and a public constructor that creates the collision.

Now, open the **TileMap** class so that we can add the collision layer to it. First, we need to add a field to the class to hold the collision layer. Update the Field Region to the following.

```
List<Tileset> tilesets;
List<ILayer> mapLayers;
CollisionLayer collisionLayer;
static int mapWidth;
static int mapHeight;
```

We want to expose the collision layer to other classes as well so add the following property to the **TileMap** class.

```
public CollisionLayer CollisionLayer
{
    get { return collisionLayer; }
}
```

Next, the two constructors need to be updated. In one constructor we will add a parameter for a collision layer and in the second we will just create a new object. Update the constructors for **TileMap** to the following.

```
public TileMap(List<Tileset> tilesets, MapLayer baseLayer, MapLayer buildingLayer, MapLayer
    splatterLayer, CollisionLayer collisionLayer)
{
    this.tilesets = tilesets;
    this.mapLayers = new List<ILayer>();
    this.collisionLayer = collisionLayer;

    mapLayers.Add(baseLayer);

    AddLayer(buildingLayer);
    AddLayer(splatterLayer);

    mapWidth = baseLayer.Width;
    mapHeight = baseLayer.Height;
```

```

}

public TileMap(Tileset tileset, MapLayer baseLayer)
{
    tilesets = new List<Tileset>
    {
        tileset
    };

    collisionLayer = new CollisionLayer();

    mapLayers = new List<ILayer>
    {
        baseLayer
    };

    mapWidth = baseLayer.Width;
    mapHeight = baseLayer.Height;
}

```

Not much out of the ordinary here, just creating a field and making sure that it get initialized. We will also want to update the **MapData** class in **WorldData** that is used for reading and writing maps for the game. Open that class and update the code to the following.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using MGRpgLibrary.TileEngine;

namespace RpgLibrary.WorldClasses
{
    public class MapData
    {
        public string MapName;
        public MapLayerData[] Layers;
        public TilesetData[] Tilesets;
        public CollisionLayer Collisions;

        private MapData()
        {
        }

        public MapData(string mapName, List<TilesetData> tilesets, List<MapLayerData> layers,
            CollisionLayer collisionLayer)
        {
            MapName = mapName;
            Tilesets = tilesets.ToArray();
            Layers = layers.ToArray();
            Collisions = collisionLayer;
        }
    }
}

```

Very similar to **TileMap**. I just added a field to store the collision layer and update the constructor to take a collision layer as a parameter and then set the collision layer.

The next change will be to the **World** class. Expand the **WorldClasses** folder in the **MGRpgLibrary** project and open the **World** class. Add the following property to the class.

```
public TileMap CurrentMap
{
    get { return levels[currentLevel].Map;
}
```

What this is doing is exposing the current map as a read only property so that we can use it for collision detection. It will be useful later on as well for other interactions such as player to NPC conversations.

I will be implementing the collision detection in the **Player** class in **EyesOfTheDragon Components** folder. So, open **Player.cs** and replace the current **Update** method with these two methods, along with the following using statement.

```
using EyesOfTheDragon.GameScreens;

public void Update(GameTime gameTime)
{
    camera.Update(gameTime);
    Sprite.Update(gameTime);

    if (InputHandler.KeyReleased(Keys.PageUp) ||
        InputHandler.ButtonReleased(Buttons.LeftShoulder, PlayerIndex.One))
    {
        camera.ZoomIn();

        if (camera.CameraMode == CameraMode.Follow)
        {
            camera.LockToSprite(Sprite);
        }
    }
    else if (InputHandler.KeyReleased(Keys.PageDown) ||
        InputHandler.ButtonReleased(Buttons.RightShoulder, PlayerIndex.One))
    {
        camera.ZoomOut();

        if (camera.CameraMode == CameraMode.Follow)
        {
            camera.LockToSprite(Sprite);
        }
    }

    Vector2 motion = new Vector2();

    if (InputHandler.KeyDown(Keys.W) ||
        InputHandler.ButtonDown(Buttons.LeftThumbstickUp, PlayerIndex.One))
    {
        Sprite.CurrentAnimation = AnimationKey.Up;
        motion.Y = -1;
    }
    else if (InputHandler.KeyDown(Keys.S) ||
        InputHandler.ButtonDown(Buttons.LeftThumbstickDown, PlayerIndex.One))
    {
        Sprite.CurrentAnimation = AnimationKey.Down;
    }
}
```

```

        motion.Y = 1;
    }

    if (InputHandler.KeyDown(Keys.A) ||
        InputHandler.ButtonDown(Buttons.LeftThumbstickLeft, PlayerIndex.One))
    {
        Sprite.CurrentAnimation = AnimationKey.Left;
        motion.X = -1;
    }
    else if (InputHandler.KeyDown(Keys.D) ||
        InputHandler.ButtonDown(Buttons.LeftThumbstickRight, PlayerIndex.One))
    {
        Sprite.CurrentAnimation = AnimationKey.Right;
        motion.X = 1;
    }

    if (motion != Vector2.Zero)
    {
        UpdatePosition(gameTime, motion);
    }
    else
    {
        Sprite.IsAnimating = false;
    }

    if (InputHandler.KeyReleased(Keys.F) ||
        InputHandler.ButtonReleased(Buttons.RightStick, PlayerIndex.One))
    {
        camera.ToggleCameraMode();

        if (camera.CameraMode == CameraMode.Follow)
        {
            camera.LockToSprite(Sprite);
        }
    }

    if (camera.CameraMode != CameraMode.Follow)
    {
        if (InputHandler.KeyReleased(Keys.C) ||
            InputHandler.ButtonReleased(Buttons.LeftStick, PlayerIndex.One))
        {
            camera.LockToSprite(Sprite);
        }
    }
}

private void UpdatePosition(GameTime gameTime, Vector2 motion)
{
    Sprite.IsAnimating = true;
    motion.Normalize();

    Vector2 distance = motion * Sprite.Speed * (float)gameTime.ElapsedGameTime.TotalSeconds;
    Vector2 next = distance + Sprite.Position;

    Rectangle playerRect = new Rectangle(
        (int)next.X + CollisionLayer.CollisionRadius,
        (int)next.Y + CollisionLayer.CollisionRadius,
        Engine.TileWidth - CollisionLayer.CollisionRadius,
        Engine.TileHeight - CollisionLayer.CollisionRadius);

```

```

foreach (Point p in GameplayScreen.World.CurrentMap.CollisionLayer.Collisions.Keys)
{
    Rectangle r = new Rectangle(
        p.X * Engine.TileWidth + CollisionLayer.CollisionRadius,
        p.Y * Engine.TileHeight + CollisionLayer.CollisionRadius,
        Engine.TileWidth - CollisionLayer.CollisionRadius,
        Engine.TileHeight - CollisionLayer.CollisionRadius);

    if (r.Intersects(playerRect))
    {
        return;
    }
}

Sprite.Position = next;
Sprite.LockToMap();

if (camera.CameraMode == CameraMode.Follow)
{
    camera.LockToSprite(Sprite);
}
}

```

What I did here is extract the logic that updates the position of the player's sprite into a new method. What the new code does is first calculates the distance the sprite will travel based on the player's input. It then adds that to the sprite's current position to get the desired destination.

The interesting code is I create a new rectangle object based on this new position. I “shrink” the rectangle by adding the collision radius to the **X** and **Y** coordinates and subtracting it from the Width and Height properties. Next there is a foreach loop that iterates over each Point that was added to the collision map. In the same way as I did for the player's sprite I create a shrunk rectangle for that tile. Since the points are tiles I had to multiple the **X** and **Y** by **TileWidth** and **TileHeight** to get the pixel representation. I then check to see if the two rectangles intersect. If they do I exit the function as we don't want the player's sprite to update. When the loop finishes I update the sprite's position, lock it to the map and then apply the camera mode logic.

What is next then is to add some tile collisions to the map to verify that this works as expected. First, open the **CharacterGeneratorScreen** class in the **GameScreens** folder of the main project. Find the **CreateWorld** method and update it to the following.

```

private void CreateWorld()
{
    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
    Tilesheet tilesheet1 = new Tilesheet(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");

    Tilesheet tilesheet2 = new Tilesheet(tilesetTexture, 8, 8, 32, 32);
    MapLayer layer = new MapLayer(100, 100);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);

```

```

        layer.SetTile(x, y, tile);
    }
}

MapLayer splatter = new MapLayer(100, 100);
Random random = new Random();

for (int i = 0; i < 100; i++)
{
    int x = random.Next(0, 100);
    int y = random.Next(0, 100);
    int index = random.Next(2, 14);

    Tile tile = new Tile(index, 0);

    splatter.SetTile(x, y, tile);
}

splatter.SetTile(1, 0, new Tile(0, 1));
splatter.SetTile(2, 0, new Tile(2, 1));
splatter.SetTile(3, 0, new Tile(0, 1));

TileMap map = new TileMap(tileset1, layer);

map.AddTileset(tileset2);
map.AddLayer(splatter);
map.CollisionLayer.Collisions.Add(new Point(1, 0), CollisionType.Impassable);
map.CollisionLayer.Collisions.Add(new Point(3, 0), CollisionType.Impassable);

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;

GamePlayScreen.World = world;
}

```

I just added two impassable tiles at (1,0) and (3,0) which represent the walls that were added but left the door open so the player can walk into the door to open it, in a manner of speaking. Now open the **LoadGameScreen** class in the same folder and replace the **CreateWorld** method with this code.

```
private void CreateWorld()
```

```

{
    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
    Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");

    Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);
    MapLayer layer = new MapLayer(100, 100);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);

            layer.SetTile(x, y, tile);
        }
    }

    MapLayer splatter = new MapLayer(100, 100);
    Random random = new Random();

    for (int i = 0; i < 100; i++)
    {
        int x = random.Next(0, 100);
        int y = random.Next(0, 100);
        int index = random.Next(2, 14);

        Tile tile = new Tile(index, 0);

        splatter.SetTile(x, y, tile);
    }

    splatter.SetTile(1, 0, new Tile(0, 1));
    splatter.SetTile(2, 0, new Tile(2, 1));
    splatter.SetTile(3, 0, new Tile(0, 1));

    TileMap map = new TileMap(tileset1, layer);

    map.AddTileset(tileset2);
    map.AddLayer(splatter);
    map.CollisionLayer.Collisions.Add(new Point(1, 0), CollisionType.Impassable);
    map.CollisionLayer.Collisions.Add(new Point(3, 0), CollisionType.Impassable);

    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;

GamePlayScreen.World = world;
}

```

There is one other thing to cover in this tutorial. That is in the level editor. Adding another parameter to the constructor broke the **saveLevelToolStripMenuItem_Click** method in the **FormMain** of the **XLevelEditor** project. The call to creating a new **MapData** object requires a **CollisionLayer** object. Update that method to the following.

```

void saveLevelToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (map == null)
        return;

    List<MapLayerData> mapLayerData = new List<MapLayerData>();

    for (int i = 0; i < clbLayers.Items.Count; i++)
    {
        if (layers[i] is MapLayer)
        {
            MapLayerData data = new MapLayerData(
                clbLayers.Items[i].ToString(),
                ((MapLayer)layers[i]).Width,
                ((MapLayer)layers[i]).Height);

            for (int y = 0; y < ((MapLayer)layers[i]).Height; y++)
                for (int x = 0; x < ((MapLayer)layers[i]).Width; x++)
                    data.SetTile(
                        x,
                        y,
                        ((MapLayer)layers[i]).GetTile(x, y).TileIndex,
                        ((MapLayer)layers[i]).GetTile(x, y).Tileset);

            mapLayerData.Add(data);
        }
    }

    MapData mapData = new MapData(levelData.MapName, tileSetData, mapLayerData, new
CollisionLayer());

    FolderBrowserDialog fbDialog = new FolderBrowserDialog();
    fbDialog.Description = "Select Game Folder";
    fbDialog.SelectedPath = Application.StartupPath;

    DialogResult result = fbDialog.ShowDialog();

    if (result == DialogResult.OK)
    {
        if (!File.Exists(fbDialog.SelectedPath + @"\Game.xml"))
        {
            MessageBox.Show("Game not found", "Error");
            return;
        }
    }
}

```

```

string LevelPath = Path.Combine(fbDialog.SelectedPath, @"Levels\");
string MapPath = Path.Combine(LevelPath, @"Maps\");

if (!Directory.Exists(LevelPath))
    Directory.CreateDirectory(LevelPath);

if (!Directory.Exists(MapPath))
    Directory.CreateDirectory(MapPath);

XnaSerializer.Serialize<LevelData>(LevelPath + levelData.LevelName + ".xml",
levelData);
XnaSerializer.Serialize<MapData>(MapPath + mapData.MapName + ".xml", mapData);
}
}

```

That's it for this tutorial. It covers the original tutorial with a few modifications. So, I encourage you to visit my blog at, <https://cynthiamcmahon.ca/blog/>, for the latest news on my tutorials and other goodness.

Good luck in your Game Programming Adventures!

Cynthia