A Nordic Blue Salmon

Me.

Not yesterday

1 Introduction

I make nordic blue salmon.

2 Mathematical Model of the Fish

2.1 Chain Class for the Fish Spine

The `Chain` class represents the backbone or spine of the fish. It models a sequence of connected segments, where each segment is defined by its length and orientation. The purpose of this class is to maintain the specified segment lengths while adjusting the chain to reach a target position.

2.2 Code Explanation of the Chain Class

Summary of it:

2.2.1 Constructor

The constructor initializes the chain with a specified number of segments and segment length:

```
Chain(Vector2f origin, int numSegments, float segmentLength)
```

Here, `origin` sets the starting position of the chain, `numSegments` defines the number of segments, and `segmentLength` specifies the length of each segment. The constructor initializes all joints to the origin position and angles to zero. Yeah you speak english you know it.

2.2.2 Direction Vector Calculation

To compute the direction vector between each pair of consecutive joints, the code uses:

```
Vector2f direction = joints[i] - joints[i - 1];
```

This line calculates the vector from joint j_{i-1} to joint j_i. The direction vector is computed as:

$$\text{direction} = \mathbf{j_i} - \mathbf{j_{i-1}}$$

where $\mathbf{j_i}$ and $\mathbf{j_{i-1}}$ are the position vectors of the joints.

2.2.3 Normalization

The direction vector is normalized to ensure it has a magnitude of 1:

```
float len = sqrt(direction.x * direction.x + direction.y * direction.y)
    ;
direction /= len;
```

This normalization is crucial for maintaining the segment length consistency. The length len of the direction vector is given by:

$$\text{len} = \sqrt{\text{direction.x}^2 + \text{direction.y}^2}$$

The normalized direction vector is:

$$\text{direction} = \frac{\text{direction}}{\text{len}}$$

2.2.4 Scaling

The normalized direction vector is scaled by the segment length:

```
direction *= segmentLength;
```

This adjustment ensures that each segment of the chain maintains the specified length. The scaled direction vector is:

$$\text{direction} = \text{direction} \times \text{segmentLength}$$

2.2.5 Updating Joint Positions

The position of each joint is updated as follows:

```
joints[i] = joints[i - 1] + direction;
```

This line sets the new position of joint j_i based on the previous joint's position and the scaled direction vector. The updated position of joint j_i is:

$$\mathbf{j_i} = \mathbf{j_{i-1}} + \text{direction}$$

2.2.6 Angle Calculation

The angle of each segment relative to the x-axis is computed using the `atan2` function: "**atan2**(y, x) returns the angle between the positive x-axis and the ray from the origin to the point (x, y), confined to (,]." WIKIPEDIA

```
angles[i] = atan2(direction.y, direction.x);
```

This angle is used to determine the orientation of each segment in the chain. The angle θ_i is computed as:

$$\theta_i = \text{atan2}(\text{direction.y}, \text{direction.x})$$

2.3 Fish Class for Full Body Representation

The `Fish` class extends the functionality of the `Chain` class to represent the entire fish, including its body and fins.

2.3.1 Body Widths and Shape

The fish's body is represented as a series of connected polygons with varying widths along the length of the fish. The width of each segment is scaled by:

$$\text{bodyWidth}[i] = \text{bodyWidths}[i] \times \text{fishScale}$$

where `bodyWidths` is an array of predefined widths, and `fishScale` is a scaling factor that adjusts the overall size of the fish.

2.3.2 Drawing the Body

To draw the fish's body, follow these steps:

1. **Calculate Vertex Positions**:

```
Vector2f getVertex(Vector2f pos, float angle, float length,
    float angleOffset = 0, float lengthOffset = 0) {
    float offsetAngle = angle + angleOffset;
    float offsetLength = length + lengthOffset;
    return Vector2f(pos.x + cos(offsetAngle) * offsetLength
        ,
                    pos.y + sin(offsetAngle) * offsetLength
                       );
}
```

This function calculates a point offset from a joint position *pos* based on its angle *angle* and the length *length*, with optional angle and length offsets.

2. **Collect Vertices**:

```
vector<Vector2f> points;
for (int i = 0; i < 10; ++i) {
    points.push_back(getVertex(j[i], a[i], bodyWidth[i],
        M_PI / 2));
}
points.push_back(getVertex(j[9], a[9], bodyWidth[9], M_PI))
    ;
for (int i = 9; i >= 0; --i) {
    points.push_back(getVertex(j[i], a[i], bodyWidth[i], -
        M_PI / 2));
}
points.push_back(getVertex(j[0], a[0], bodyWidth[0], -M_PI
    / 6));
points.push_back(getVertex(j[0], a[0], bodyWidth[0], 0));
points.push_back(getVertex(j[0], a[0], bodyWidth[0], M_PI /
    6));
```

Gather all vertices into a list, ensuring to cover both sides of each joint and close the shape.

3. **Draw the Polygon**:

```
void drawPolygon(RenderWindow& window, const vector<
    Vector2f>& points, const Color& color, float
    outlineThickness) {
    ConvexShape shape;
    shape.setPointCount(points.size());
    for (size_t i = 0; i < points.size(); ++i) {
        shape.setPoint(i, points[i]);
    }
    shape.setFillColor(color);
    shape.setOutlineColor(white);
    shape.setOutlineThickness(outlineThickness);
    window.draw(shape);
}
```

Use the collected vertices to render the polygon shape of the fish's body on the screen.

4. **Function Usage**:

```
drawPolygon(window, points, bodyColor, 2);
```

This line draws the body polygon using the calculated vertices.

2.3.3 Fins

Yeah you just add like an ellipse to the side and make

So this thing kinda uses the ‘getPos‘ function to determine the position of the fin, sets the dimensions given and color of the fin. THEN it adjusts its origin to the center, and finally SHOCKER, draws it on the window. Parameters if u can’t read

- `window`: The SFML render window where the fin will be drawn.
- `index`: The index of the joint where the fin will be attached.
- `angleOffset`: An angle offset to rotate the fin relative to its base angle.
- `width` and `height`: Dimensions of the fin.
- `rotation`: The rotation angle of the fin.

```
void drawFin(RenderWindow& window, int index, float angleOffset, float
    width, float height, float rotation) {
     Vector2f pos = getPos(index, angleOffset, 0);
     RectangleShape fin(Vector2f(width, height));
     fin.setFillColor(finColor);
     fin.setOrigin(width / 2, height / 2);
     fin.setPosition(pos);
     fin.setRotation(rotation * 180 / M_PI);
     window.draw(fin);
}
```

2.4 Explanation

3 Dependence on ‘drawFin‘ and why you need it if you want to be able to move (not in the game just it looks cool)

All other fin-drawing methods in the fish animation code rely on ‘drawFin‘ for rendering the pectoral fins. These methods use the same principles of positioning, dimensioning, and rotation to draw the fins at different parts of the fish.