Organic Curve Rendering

The Challenge

To accommodate the wide range of movement and stretching the character's main arms require, we decided to use a line renderer to represent each arm rather than a 3D model. This helped to avoid issues with the physics and animation of the character and allowed the arms to stretch and bend fluidly. The main challenge with the arms was to make the arm's movement look organic and natural, rather than a series of 2D rectangles. It also prevented problems with textures stretching on a 3D model and gave us free reign over the length and shape of the tentacle.

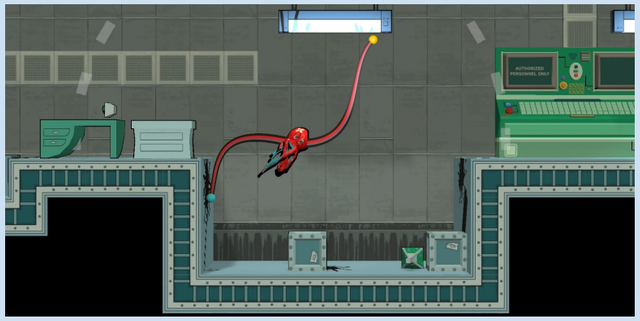
Solution

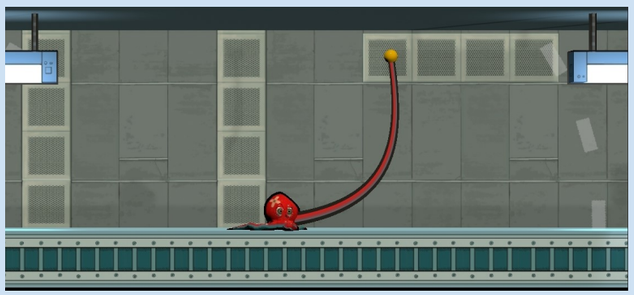
The starting point for my solution was a script to draw a cubic Bezier curve between two points, taken from http://www.theappguruz.com/blog/bezier-curve-in-games. I modified this code to draw the curve based on two control points, in addition to a beginning and end. These controls points were implemented as empty transforms on the character's arms.

In order to make the arms move more organically (rather than drawing a uniform Bezier curve) I created a CurveManager class that handles the way the curve behaves. The curve manager places the control point transforms on the arms and moves them as the arms move based on the position of the end of the arm.

When the arms are near the octopus, the control points are moved to create small, tight curves, while at longer distances the curve is less pronounced. The curve manager also checks which side of the character's body the arm is on, in order to correctly orient the arm texture and prevent the arm from bending in the wrong direction.

To accomplish this, the control points are updated each frame, with the new position found via a linear interpolation between the current position and a target position constructed from the x and y values produced by the checkCurveDirection function. This allows the arms to fluidly change between different curve shapes as the player moves the arm.





Improvements

Although I am satisfied with the current effect, there are still many improvements that could be made to the algorithm. More fine-grained conditions in checkCurveDirection could allow for a larger variety of behaviours.

Additionally, excessive movement of the character - such as when hanging from both arms, can cause the curves to move so quickly between shapes that it looks odd. Looking at solving this would make the arms look more realistic.

Pseudocode of CurveManager Class

```
controlPoint1Pos = 33
controlPoint2Pos = 66

updateControlPoints
{
      bodyToArmVector = armPosition - playerPosition

      xPosition1 = bodyToArmVector * (controlPoint1Pos / 100)
      xPosition2 = bodyToArmVector * (controlPoint2Pos / 100)

      checkCurveDirection() //Calculates y positions

      controlPoint1Target = new vector using x and y positions
      controlPoint2Target = new vector using x and y positions

      lerpPos1 = lerp between current position and target position over deltaTime
      lerpPos2 = lerp between current position and target position over deltaTime

      controlPoint1Pos = lerpPos1
      controlPoint2Pos = lerpPos2
}

checkCurveDirection
{
      Sign = 1
      Set lineRenderer texture

      If bodyToArmVector.x < 0
            Sign = -1
            Flip lineRenderer texture

      If bodyToArmVector.x <= smallCurveDistance
            If bodyToArmVector.y >= 0
                  //Do small upwards curve
                  Set Y positions

            Else if bodyToArmVector.y < 0
                  //Do small downwards curve
                  Set Y positions

      If bodyToArmVector.y >= 0
            //Do long upwards curve
            Set Y positions

      Else if bodyToArmVector.y < 0
            //Do long downwards curve
            Set Y positions
}
```