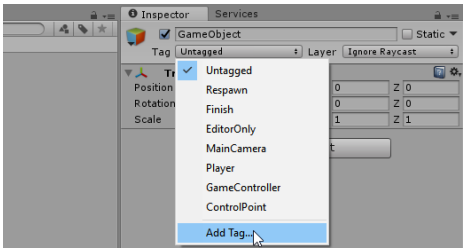
1 Creating a Unity Project

In this activity we will creating a Unity project that can draw b-spline curves. Start by creating a new 2D unity project. Note that spaces have been added to the code for this activity to allow the lines to wrap more neatly.

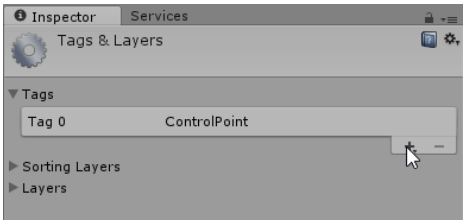
2 Creating a Control Point

We will start by creating a single Control Point that we will turn into a prefab and use to dynamically create the desired number of Control Points.

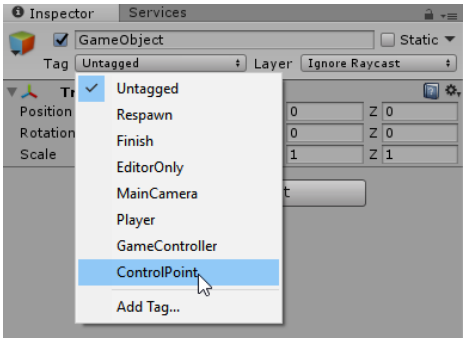
Create a new sprite by selecting GameObject > 2D Object > Sprite and call it "Control Point". In order to get a round shaped control point that is nicely sized set the sprite to "Knob" and set the scale to (2, 2, 1). So that the Control-Point can be clicked add the "Box Collider 2D" component to the GameObject and ensure that the "Is Trigger" Checkbox is ticked. We also need to set the tag of the ControlPoint GameObject to be "ControlPoint". This can be done by selecting "Add Tag" from the Tag dropdown menu.



Next Press the plus button under tags and name the tag "ControlPoint".



Lastly we can add the tag to the GameObject by clicking on the tag in the Tag dropdow menu.



Now that the Control Point GameObject has been set up we can create a new C# script called "ControlPoint". This script will only handle the moving of control points. At the top of the script add a new public boolean called "isMoving". We will use this to determine if the Control Point should follow the mouse.

```
        public bool isMoving = false;
```

Now within the Update method we should create a new if statement to test if the Control Point should be moving.

```
        // Update is called once per frame
        void Update () {

                if (isMoving) {
```

Within that if statement we should then create a variable to store the new position of the Control point.

```
                        Vector3 position = new Vector3();
```

Next we should get the mouse position in world space.

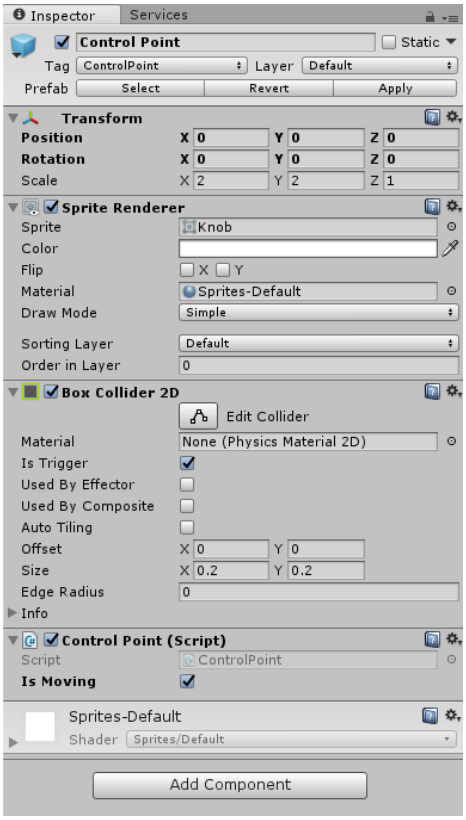
```
                        // Find the mouse position in word space
                        Vector2 mousePosition =
                            Camera.main.ScreenToWorldPoint(
                            Input.mousePosition);
```

We can then set the new position of the GameObject to be equal to the x and y coordinates of the mouse in world space.

```
// Find the new position of the gameobject
position.x = mousePosition.x;
position.y = mousePosition.y;
position.z = transform.position.z;

transform.position = position;
```

Now we can add the ControlPoint script to the Control Point GameObject. In order to test the functionality of the script we can then tick the "Is Moving" Checkbox and the control point should begin to follow the mouse.



The Control Point is now ready to be turned into a prefab. Create a Prefabs folder then, ensure that the "Is Moving" checkbox is unchecked, and drag the GameObject into your Prefabs folder. You can now delete the Control Point GameObject from the hierarchy if you wish to do so.

3 Drawing the B-spline

Now that the Control Points are working correctly we can begin working on the b-spline itself. Create a new empty GameObject and call it "B-spline". You should then add a Line Renderer to the B-Spline GameObject.

We can now start working on the code to draw the b-spline. Create a new C# script and call it "Bspline". At the top of this script we then need to create some public variables that we will use to set the number of control points that we will start with, the number of line positions that will be used per control point, the order of the curve, the controlPoint prefeb and the width of the curve, and some private variables that we will use when calculating where each of the line positions will be placed.

```
        // Spline variables
        public int numControlPoints = 9;
        public int numLinePositionsMultiplier = 10;
        public int k = 4;

        private int numLinePositions;
        private int m;
        private int n;
        private List<float> knots;

        public GameObject controlPoint;

        private List<GameObject> controlPoints;

        // Linerenderer variables
        public float lineWidth = 0.05f;
```

We can now start working on the "SetupBspline" method. We will use this method to calculate many of the values that need to be calculated when the program is initially run, and a new Control Point is added.

```
        // Set the bspline variables
        void SetupBspline () {
```

Inside this method we will first calculate the values of m, n and numLinePositions.

```
                m = numControlPoints - 1;
                n = m + k;

                numLinePositions = numControlPoints *
                    numLinePositionsMultiplier;
```

Now that the "SetupBspline" method has been created we can start working on the Start method. Inside the Start method first call the "SetupBspline" method.

```
// Set the b-spline variables
SetupBspline();
```

Next we will calculate the starting position of the control points so that they form a sine curve and spawn the control points at these locations.

```
// Spawn the control points
GameObject[] initialControlPoints = new
    GameObject[numControlPoints];

float intervalSize =
    Camera.main.ScreenToWorldPoint(new
    Vector2(Screen.width, 0.0f)).x / numControlPoints;
for(int i = 0; i < numControlPoints; i++) {
        initialControlPoints[i] =
            Instantiate(controlPoint, this.transform);

        // Set initial position
        float startPosition =
            Camera.main.ScreenToWorldPoint(new
            Vector2(0.0f, 0.0f)).x / 2;
        startPosition += intervalSize / 2;
        float xPosition = startPosition +
            (intervalSize * i);
        float yPosition = Mathf.Sin(xPosition);
        initialControlPoints[i].transform.position =
            new Vector3(xPosition, yPosition, 1.0f);
}

controlPoints = new
    List<GameObject>(initialControlPoints);
```

If we set the value of the "Control Point" variable to be our control point prefab, the control points should now be able to be spawned into the scene in appear similar to the following:

We can now start working on the "MoveControlPoint" method. This method will be used to move a control point when it is clicked.

```
        // Move control points
        void MoveControlPoint () {
```

If the mouse1 button is pressed when the mouse is over one of the control points the "isMoving" variable of that control point should be set to true.

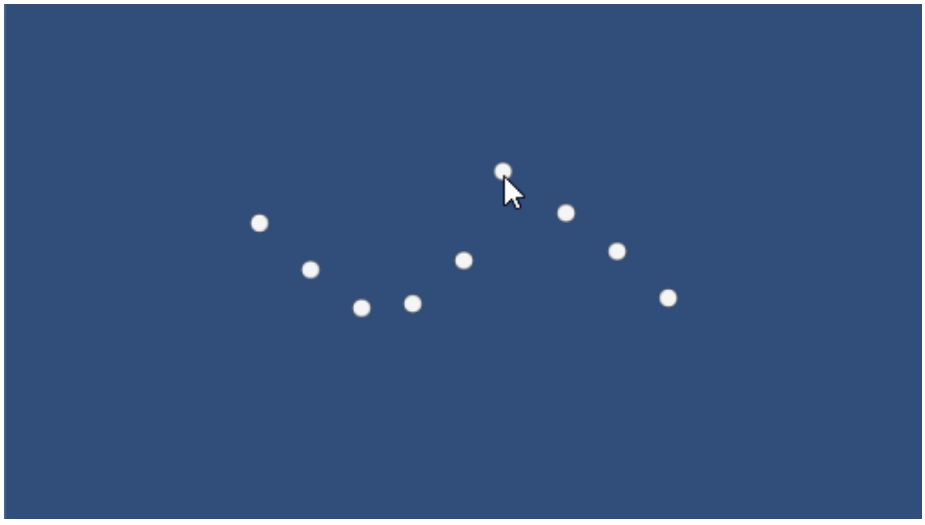
```
                // Hold the control point
                if (Input.GetMouseButtonDown(0)) {
                        Vector2 mousePosition =
                            Camera.main.ScreenToWorldPoint(
                            Input.mousePosition );

                        Collider2D hitCollider =
                            Physics2D.OverlapPoint(mousePosition);
                        if (hitCollider && hitCollider.transform.tag
                            == "ControlPoint"){
                                hitCollider.transform.gameObject.
                                    GetComponent<ControlPoint>().
                                    isMoving = true;
                        }
```

Now we can pick up control points but not place them again. In order to do so can set the value of "isMoving" to false for all of the control points.

```
            } else if (Input.GetMouseButtonUp(0)) {
                    // Drop all controlpoints
                    foreach (GameObject controlPoint in
                        controlPoints) {
                            controlPoint.GetComponent<ControlPoint>().
                                isMoving = false;
                    }
            }
```

If we now call the "MoveControlPoint" method from the Update method the control points should follow the mouse when the left mouse button is held down, and stop following when the left mouse button is released.



Next we can start working on the "AddOrRemoveControlPoint" method. This method will add a new control point if the right mouse button is pressed in an empty space, and remove a control point is the right mouse button is pressed over a control point.

```
        // Add new controlpoint
        void AddOrRemoveControlPoint () {
```

Inside this method we should first create a new if statement to test if the right mouse button has been pressed.

```
                if (Input.GetMouseButtonDown(1)) {
```

We should then find the position of the mouse in world space and find any GameObjects that the mouse is over.

```
                        Vector2 mousePosition =
                            Camera.main.ScreenToWorldPoint(
                            Input.mousePosition);

                        Collider2D hitCollider =
                            Physics2D.OverlapPoint(mousePosition);
```

If there is a collider under the mouse and the tag of the collider is "Control-Point", we destroy the GameObject, remove it from the list of control points and decrement the number of control points.

```
                        if (hitCollider && hitCollider.transform.tag
                            == "ControlPoint"){
                                numControlPoints -= 1;
                                controlPoints.Remove(
                                    hitCollider.transform.gameObject);
                                Destroy(
                                    hitCollider.transform.gameObject);
```

Otherwise we should create a new Control point at the mouse's current location.

```
                        } else {
                                GameObject newControlPoint =
                                    Instantiate(controlPoint,
                                    this.transform);

                                // Set the position of the new
                                    controlPoint
                                Vector3 position = new Vector3();

                                position.x = mousePosition.x;
                                position.y = mousePosition.y;
                                position.z = transform.position.z;

                                newControlPoint.transform.position =
                                    position;
```

```
                numControlPoints += 1;
                controlPoints.Add(newControlPoint);
            }
```

Lastly we should call "SetupBspline" inside the outermost if statement to ensure that the all of the variables have the correct values.

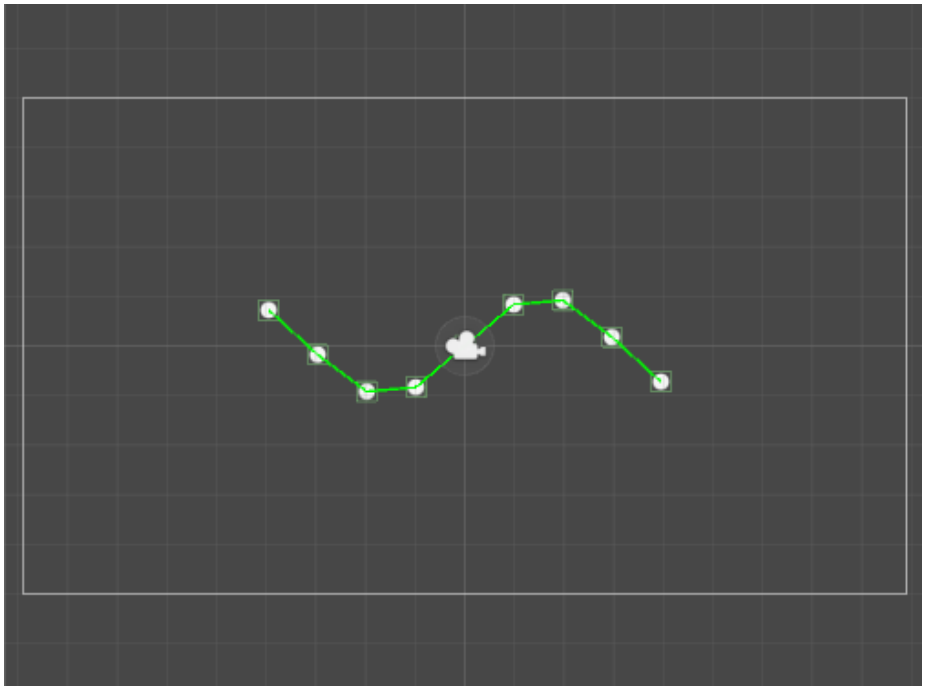
```
            SetupBspline();
```

We can now call the "AddOrRemoveControlPoint" method from the Update method. Right clicking in the scene should now remove a control point if the mouse is over a control point and add a control point if the mouse is over an empty area.

Next we should create the "DrawDebugLines" method. This method will draw lines that can only be seen in the scene editor.

```
        // Draw debug lines between control points
        void DrawDebugLines () {
                for (int i = 0; i < controlPoints.Count - 1; i++) {
                        Debug.DrawLine(controlPoints[i].transform.position,
                            controlPoints[i + 1].transform.position,
                            Color.green);
                }
        }
```

If we now call the "DrawDebugLines" method from the Update method we will be able to see lines that are drawn between each of the control points and the control points before and after it.



Now we can start actually drawing the curve. We will be doing so using "de Boor's Algorithm". Our implementation will be based on the webapp found here. Start by creating a method called "DrawCurve". This method will setup the Line Renderer and then use some helper methods to find the each of the positions of the line.

```
        // Draw the b-spline
        void DrawCurve () {
                LineRenderer curve = GetComponent<LineRenderer>();
                curve.widthMultiplier = lineWidth;
                curve.positionCount = numLinePositions;

                for (int i = 0; i < numLinePositions; i++) {
                        curve.SetPosition(i, FindBspline(knots[k - 1]
                            + (knots[m + 1] - knots[k - 1]) * i /
                            (numLinePositions - 1)));
                }
        }
```

Next we can create our first helper method, "FindBspline", which we will use to find the x and y components of each position using our second helper method.

```
        private Vector3 FindBspline (float t) {
                int i = k - 1;

                while (knots[i + 1] < t) {
                        i++;
                }

                if (i > m) {
                        i = m;
                }

                float x = BsplineBasis(i - 3, k, t) * controlPoints[i
                    - 3].transform.position.x + BsplineBasis(i - 2,
                    k, t) * controlPoints[i - 2].transform.position.x
                    + BsplineBasis(i - 1, k, t) * controlPoints[i -
                    1].transform.position.x + BsplineBasis(i, k, t) *
                    controlPoints[i].transform.position.x;
                float y = BsplineBasis(i - 3, k, t) * controlPoints[i
                    - 3].transform.position.y + BsplineBasis(i - 2,
                    k, t) * controlPoints[i - 2].transform.position.y
                    + BsplineBasis(i - 1, k, t) * controlPoints[i -
                    1].transform.position.y + BsplineBasis(i, k, t) *
                    controlPoints[i].transform.position.y;

                return new Vector3(x, y, 1.0f);
        }
```

And finally our second helper method which we will use to recursively find each component.

```
        private float BsplineBasis(int i, int k, float t) {
                if (k == 1) {
                        if ((knots[i]<=t) && (t<knots[i+1])) {
                                return 1;
                        } else {
                                return 0;
                        }
                } else {
                        return BsplineBasis(i, k - 1, t) *
                            (t-knots[i]) / (knots[i + k -
                            1]-knots[i]) + BsplineBasis(i + 1, k - 1,
                            t) * (knots[i + k] - t) / (knots[i + k] -
                            knots[i + 1]);
                }
        }
```

We can now complete our Update method my calling our "DrawCurve" if the number of control points is greater than the order of the b-spline.

```
        // Update is called once per frame
        void Update () {
                // Move the control points
                MoveControlPoint();

                // Add or remove control points
                AddOrRemoveControlPoint();

                // Draw debug lines between control points
                DrawDebugLines();

                // Recalculate the curve
                if (numControlPoints >= k) {
                        DrawCurve();
                }
```

The code should now be completed, and the output should appear similar to the following.