Heatmap Visualization Development

Locomotion System

The Locomotion System class in Unity is part of the XR (Extended Reality) toolkit, which is used to manage and implement movement and interactions in virtual reality (VR) and augmented reality (AR) environments.

Access Positional and Rotational Data

```
public InputActionProperty leftHandSnapTurnAction
{
    get => m_LeftHandSnapTurnAction;
    set => SetInputActionProperty(ref m_LeftHandSnapTurnAction, value);
}
public InputActionProperty rightHandSnapTurnAction
{
    get => m_RightHandSnapTurnAction;
    set => SetInputActionProperty(ref m_RightHandSnapTurnAction, value);
}
```

1. These properties provide get and set access to the input actions for both hands. The setter uses a custom method to properly update the input action properties.

```
protected void OnEnable()
{
    m_LeftHandSnapTurnAction.EnableDirectAction();
    m_RightHandSnapTurnAction.EnableDirectAction();
}
protected void OnDisable()
{
    m_LeftHandSnapTurnAction.DisableDirectAction();
    m_RightHandSnapTurnAction.DisableDirectAction();
}
```

2. Here, they enable or disable direct handling of the input actions for snap turns, ensuring that the actions are only active when the Game Object is active.

Heatmap Visualization Development

```
protected override Vector2 ReadInput()
{
    var leftHandValue = m_LeftHandSnapTurnAction.action?.ReadValue<Vector2>() ?? Ve
    var rightHandValue = m_RightHandSnapTurnAction.action?.ReadValue<Vector2>() ??

    return leftHandValue + rightHandValue;
}
```

3. This method overrides a base class method to read the input values from both controllers. It attempts to read the Vector2 value from each controller's action. If the action is null, it defaults to Vector2.zero. The inputs from both hands are then combined.

Logic for Implementing Heatmap (Particle System Approach)

Step 1: Define Your Data Structure
First, define the data structure for storing heatmap data points. Each point might have a position and a value indicating the intensity.

Step 2: Create the Heatmap Generator Script
Create a new C# script named Heatmap Generator and open it for editing. This script will be responsible for controlling the particle system and emitting particles based on the heatmap data.

Step 3: Attach and Configure the Script

1. Attach the Heatmap Generator script to a Game Object in your Unity scene, ideally the same Game Object as your Particle System.
2. Drag and drop your Particle System component onto the heatmap Particle System public field in the Heatmap Generator script via the Inspector.

Step 4: Generating Sample Data
You can test this system by creating sample data points and invoking the EmitHeatmapData method either from another script or by adding a method call in Start or Update for testing purposes.

Script

```
using UnityEngine;

// Struct to hold heatmap data
public struct HeatmapDataPoint
{
public Vector3 position; // Position of the data point
public float intensity; // Normalized intensity (0-1) of the data point

public HeatmapDataPoint(Vector3 pos, float inten)
{
```

Heatmap Visualization Development

```
position = pos;
intensity = inten;
}
}

public class HeatmapGenerator : MonoBehaviour
{
public ParticleSystem heatmapParticleSystem; // Reference to the Particle System component
attached to the same GameObject

// Function to emit particles based on provided data points
public void EmitHeatmapData(System.Collections.Generic.List<HeatmapDataPoint> dataPoints)
{
ParticleSystem.EmitParams emitParams = new ParticleSystem.EmitParams();

// Loop through each data point in the provided list
foreach (var dataPoint in dataPoints)
{
// Set the position of the particle to the data point's position
emitParams.position = dataPoint.position;
// Set the size of the particle based on the intensity of the data point
emitParams.startSize = Mathf.Lerp(0.1f, 2.0f, dataPoint.intensity);
// Set the color of the particle from blue (low intensity) to red (high intensity) based on the intensity
emitParams.startColor = Color.Lerp(Color.blue, Color.red, dataPoint.intensity);

// Emit one particle per data point using the parameters set above
heatmapParticleSystem.Emit(emitParams, 1);
}
}

// Example usage to generate and emit data points
void Start()
{
System.Collections.Generic.List<HeatmapDataPoint> exampleDataPoints = new
System.Collections.Generic.List<HeatmapDataPoint>
{
new HeatmapDataPoint(new Vector3(0, 0, 0), 0.1f), // Low intensity
new HeatmapDataPoint(new Vector3(1, 0, 0), 0.5f), // Medium intensity
new HeatmapDataPoint(new Vector3(2, 0, 0), 1.0f)  // High intensity
};

// Call to emit particles based on the example data points above
EmitHeatmapData(exampleDataPoints);
}
}
```

Heatmap Visualization Development

Implementation Process

1. Make sure that 3D scene has already been created.
2. In the Unity Editor, right-click in the Hierarchy panel, select Effects > Particle System.
3. Named the Game Object "HeatmapParticleSystem".
4. Creation of Script ->
    - Create a C# script in the assets folder.
    - Name that script “HeatmapGenerator”.
    - Drag this script to the "HeatmapParticleSystem" game object.
5. Select the Particle System Game Object.
6. Disable the looping system.
7. Set colour gradient according to you (blue – low intensity, red – high intensity)
8. Play the visualization.

Technical Decision

The particle system approach for creating heatmaps in Unity leverages Unity's built-in Particle System, traditionally used for visual effects like smoke, fire.

So, I have used this in one of the demo scene.
I want to show that one of the room of that demo scene has catches the fire.

Properties: -

1. Duration – 10 sec
2. Looping - disabled
3. Start delay – 0 sec.
4. Start lifetime – 5 sec (after 5 sec particles will start fading)
5. Start Size – 2.5
6. Start colour – FFFBF9.
7. Max particles – 1000
8. Shape – Cone
9. Gravity – 0.2
10. Colour over Lifetime – Blue (0E3CEC) – low Intensity, Red (F24B18) – high intensity

Heatmap Visualization Development

Images