**Experiment Report - 83 - test20\_SmoothFollow**

1. **Summary Table of Errors Found**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Error ID | Line Number | Error Type | Self-Detected? | Peer 1 Found? | Peer 2 Found? |
| E01 | line 13 | Semantic | √ | × | × |
| E02 | line 38 | Syntax | √ | √ | √ |
| E03 | line 49 | Logic | √ | √ | √ |
| E04 | line 90 | Logic | √ | × | × |

Additional Errors Found by Self: 0

Self-Review Detection Rate: 100%

Peer 1 Detection Rate: 50%

Peer 2 Detection Rate: 50%

1. **Source Code**
2. using UnityEngine;
3. public class SmoothFollow : MonoBehaviour {
4. // The target we are following
5. public Transform target;
6. // The distance in the x-z plane to the target
7. public float distance = 10.0f;
8. // the height we want the camera to be above the target
9. public float height = 5.0f;
10. // How much we
11. public float heightDamping = 2.0f;
12. public float rotationDamping = 2.0f;
13. //当前帧数值,用来检测是否有重复更新
14. private int curFrame = -1;
15. //是否优化动态建筑显示
16. public bool useOptimizeDynamicBuild = true;
17. //相机视截体
18. public Plane[] planes;
19. private Camera mainCamera = null;
20. public bool CameraMoving;
21. public Transform mTransform = null;
22. // Use this for initialization
23. void Start ()
24. {
25. mainCamera = Camera.main;
26. //Debug.Log("start camera name" + mainCamera.name);
27. }
28. //获取Transform
29. public Transform GetTransform()
30. {
31. if (mTransform = null)
32. {
33. mTransform = gameObject.transform;
34. }
35. return mTransform;
36. }
37. //创建视平截体
38. public void RefreshFrustumPlanes()
39. {
40. if (useOptimizeDynamicBuild)
41. return;
42. if (mainCamera == null)
43. return;
44. planes = GeometryUtility.CalculateFrustumPlanes(mainCamera);
45. }
47. //检测包围盒是否可见
48. public bool IsInFrustum(Plane[] planes,Bounds bound)
49. {
50. return GeometryUtility.TestPlanesAABB(planes, bound);
51. }
53. // Update is called once per frame
54. public void FixedUpdatePosition()
55. {
56. // Early out if we don't have a target
57. if (target == null || CameraMoving)
58. {
59. return;
60. }
61. //相机位置不随玩家位置高度，高度60固定
62. Vector3 targetPos = new Vector3(target.position.x, 60, target.position.z);
63. // Calculate the current rotation angles
64. float wantedRotationAngle = target.eulerAngles.y;
65. float wantedHeight = targetPos.y + height; //60 +？
66. float currentRotationAngle = transform.eulerAngles.y;
67. float currentHeight = transform.position.y;
68. // Damp the rotation around the y-axis
69. currentRotationAngle = Mathf.LerpAngle(currentRotationAngle, wantedRotationAngle, rotationDamping \* Time.deltaTime);
70. // Damp the height
71. currentHeight = Mathf.Lerp(currentHeight, wantedHeight, heightDamping \* Time.deltaTime);
72. // Convert the angle into a rotation
73. Quaternion wantRotation = Quaternion.Euler(0, currentRotationAngle, 0);
75. Vector3 wantPos = targetPos + wantRotation \* Vector3.forward \* distance;
76. wantPos.y = currentHeight;
77. Vector3 nowPos = GetTransform().position;
78. float mtoDis = Vector3.Distance(wantPos, nowPos);
79. GetTransform().position = wantPos;//给摄像机设置位置
80. Vector3 dir = targetPos - GetTransform().position;
81. dir.Normalize();//设置方向
82. GetTransform().rotation = Quaternion.LookRotation(dir);
84. RefreshFrustumPlanes();
85. //检测是否有重复更新
86. if (curFrame == Time.frameCount)
87. {
88. Debug.LogError("update more than once in one frame!");
89. }
90. curFrame = Time.frameCount;
91. }
92. }