Lost Instrument

1. Description

1.1 Overview

We use **Unity3D** to develop a 3D interactive system called ***Lost Instrument*** as our final *Human-Computer Interaction* course project.

Unity3D is a ultimate game development platform. Though it is mainly used to design 3D games, we can also use it to build a system that presents 3D products to users.

In our final project, we find three 3D models for three Chinese traditional musical instruments: *erhu*, *zither* and *lute* on the Internet and use 3D Studio Max to make them more beautiful . Every model is consist of small components and we distract these components from the models to make their own 3D models. In our **Unity3D** project, we import these three models and present them to users. Users can interact with our models by clicking on buttons our objects to listen to the music played by the instrument, get a closer look on the instrument, see the instrument fall apart into small parts and so on.

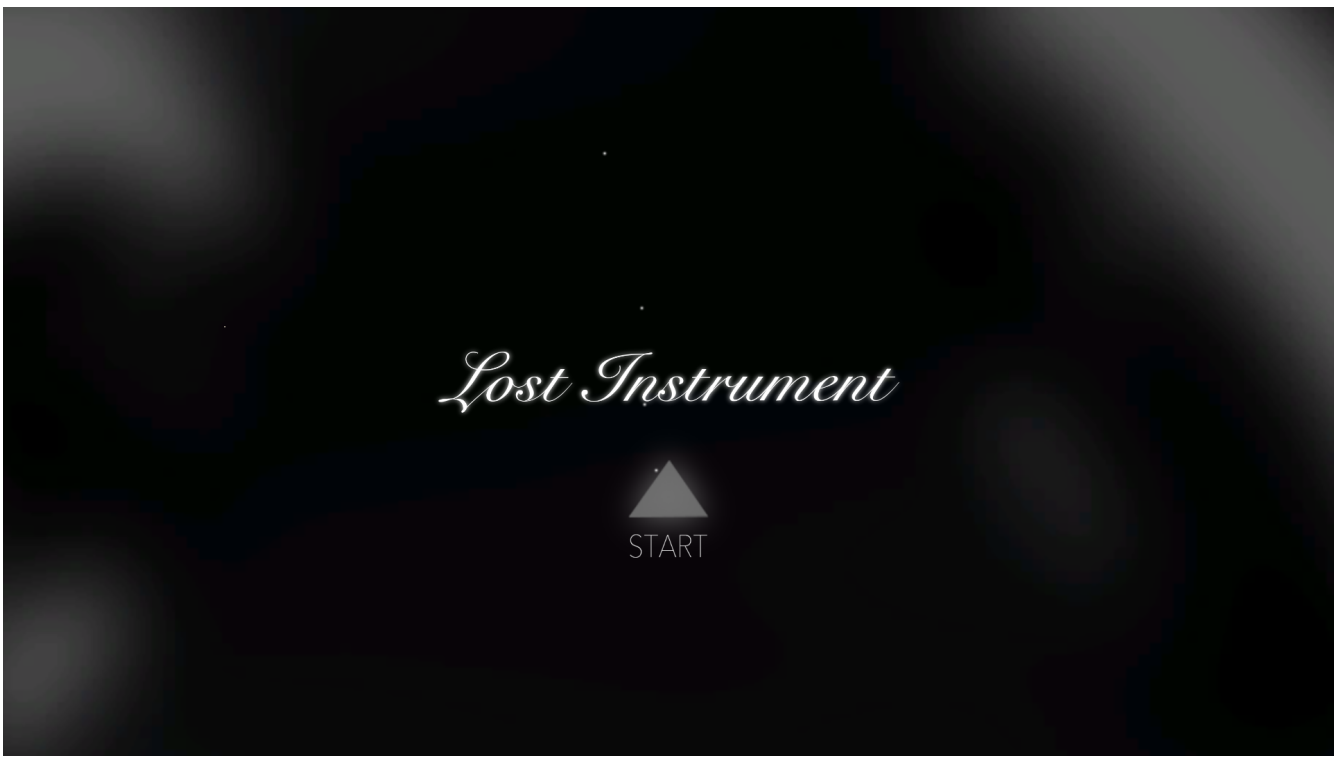
The reason why we develop this system is that Chinese traditional musical instruments are the precious wealth of the Chinese nation. Besides, as the professor told us, 3D interfaces are more natural than 2D interfaces for they are more *real*. Though there are many 3D interactive systems of western instruments, 3D interactive systems of traditional instruments are still rare. So we decided to develop a 3D interactive system of traditional instruments. We call it ***Lost Instrument*** because traditional instruments need more attention now. Our aim is to let more people know about the great treasure of our country by enabling them to use computers to interact with 3D models of traditional instruments.

1.2 Structure and Modules

We design four kinds of scenes for our project, which are **MainPage**, **chooseScene**, **instrumentScene**(including **erhuScene**, **guzhengScene**, **pipaScene**), **componentScene**(including **qingongScene**, **ganScene**, **xianScene**, **qintongScene**, **shankouScene**, **zhongxianScene**, **mianbanScene**). These scenes have corresponding **C# Scripts** for them to realize the interaction.

1.2.1 MainPage

When our user starts our program, he will first see this scene and can **click** on *START* to go to **chooseScene**.



1.2.2 chooseScene

In this scene, our user can choose instrument and he can **double-click** on the instrument to go to **instrumentScene**.

1.2.3 instrumentScene

In this Scene, our user can interact with the instrument.

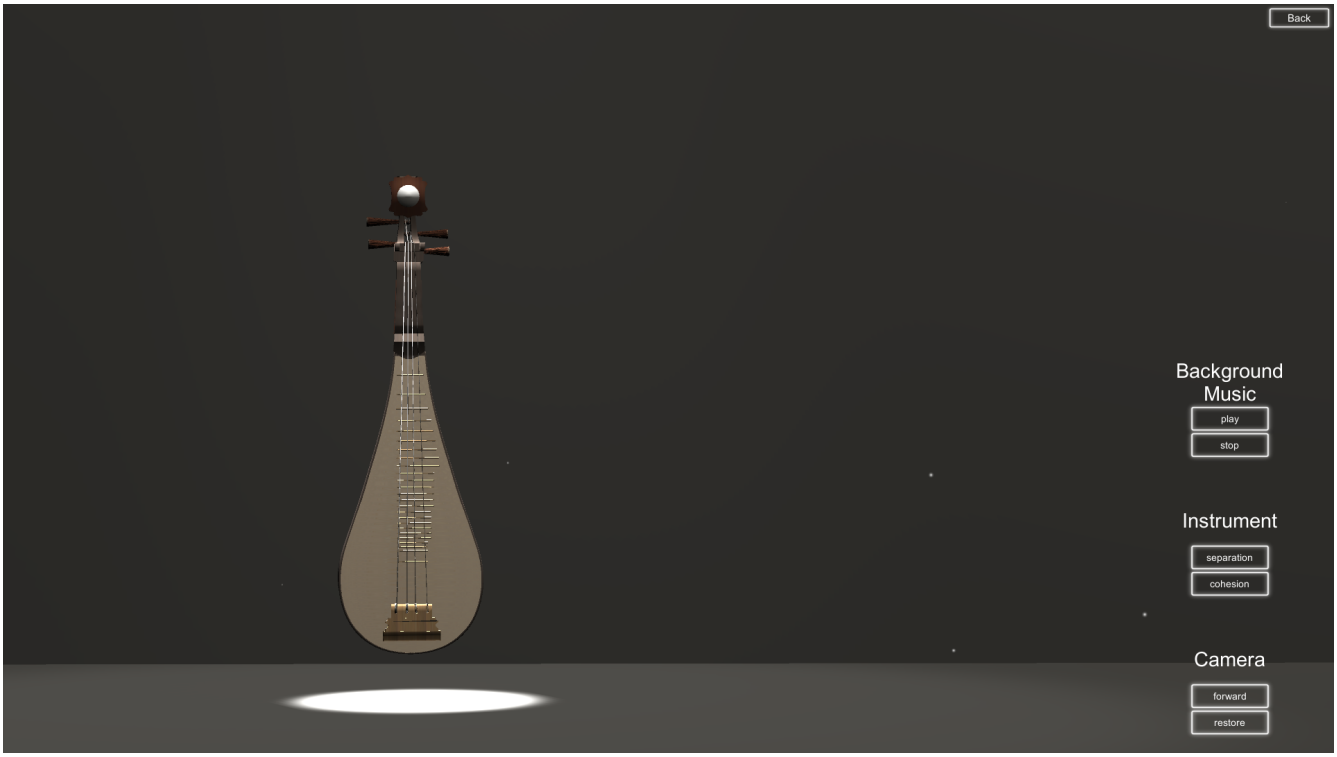
Erhu


Back
Background Music
play
stop
Instrument
separation
cohesion
Camera
forward
restore

Zither


Back
Background Music
play
stop
Instrument
separation
cohesion
Camera
forward
restore

Lute

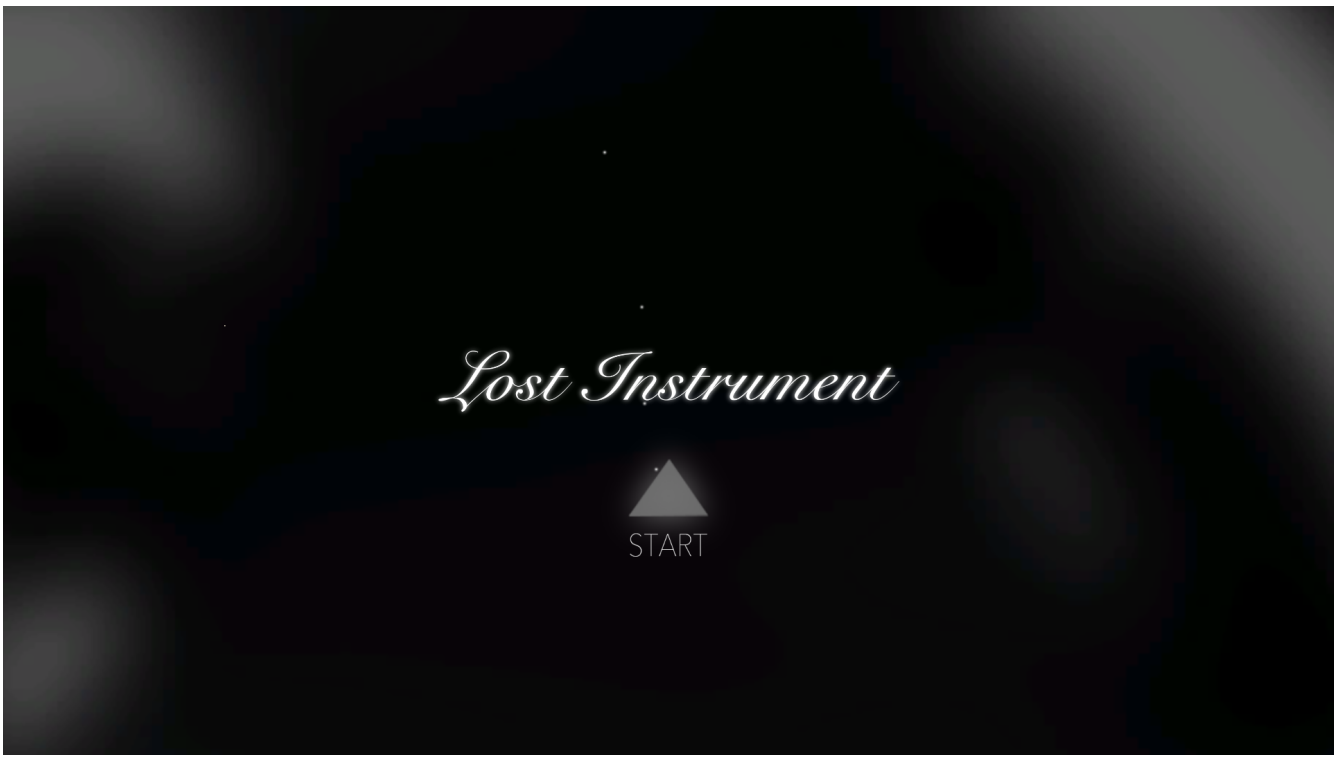


1.2.4 componentScene

In this Scene, our user can interact with the component. We create many component scenes and we only show one of them here.

2. Implemented Requirements

2.1 Interaction in MainPage



- we use Particle System for the *START* button and it has a fascinating effect.

2.2 Interaction in ChooseScene

In this scene, we have four features which are as follows.

- our user can **click** on the instrument he wants to see, and then the instrument will rotate to him. Then light will always focus on the instrument in the front.

- When our user **double-clicks** on the instrument, he can then enter the **instrumentScene**.
- When our user **double-clicks** on the 3D text *Lost Instrument*, he can return to the **MainPage**.
- We provide a *Quit* button for our user and he can **click** on it to quit our program.

2.3 Interaction in instrumentScene

In this scene, we have nine features which are as follows.

- We provide a *return* button for our user and when he **clicks** on it, he will return to the **chooseScene**.

- We provide a *play* button and a *stop* button for our user. When our user **clicks** on the *play* button, the system will play a song played with the corresponding instrument. When our user **clicks** on the *stop* button, the song will stop playing.
- We provide a *forward* button for our user and when he **clicks** on it, the camera will move forward and get more closer with the instrument.

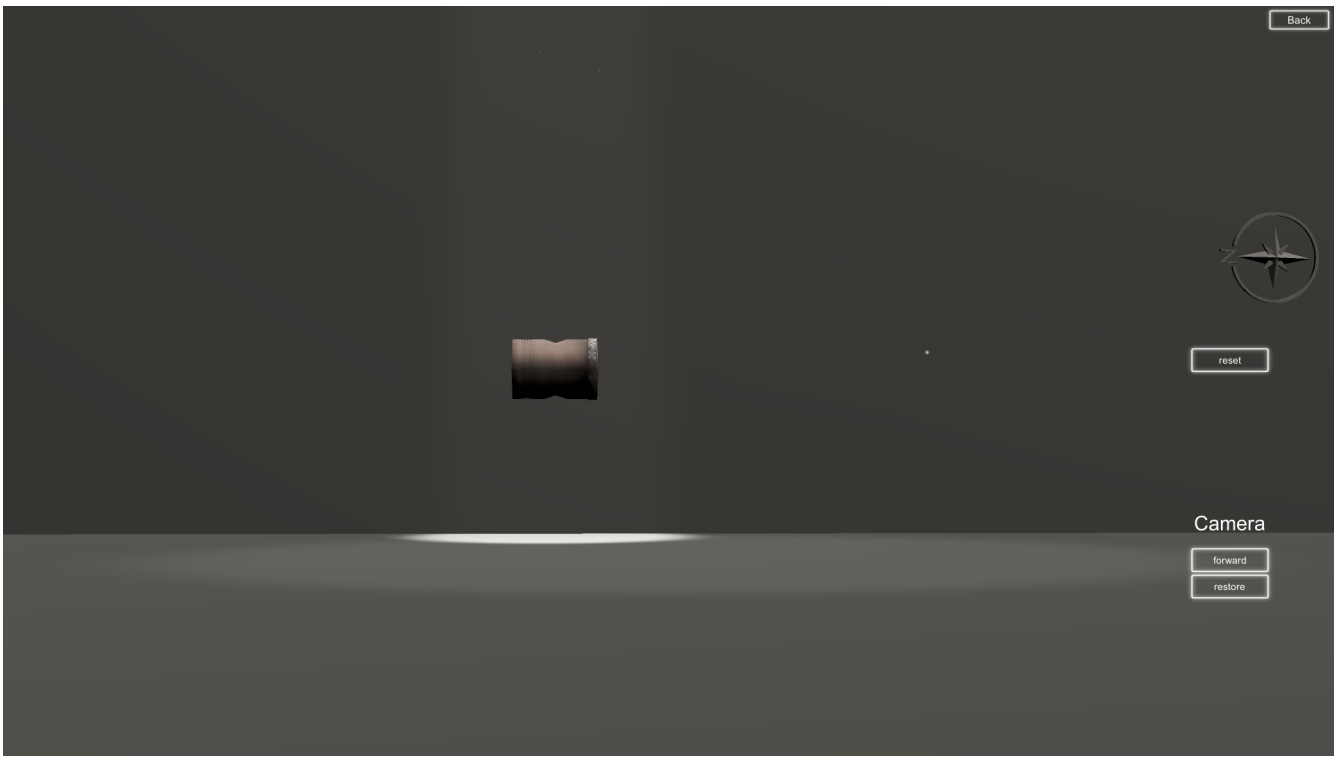
- We provide a *return* button for our user and when he **clicks** on it, the camera will return to its origin place.
- We provide a *separation* button and a *cohesion* button for our user. When our user **clicks** on the *separation* button, the instrument will separate into some components. When our user **clicks** on the *cohesion* button, the separate components will get together again.

- When our user **clicks** on a component, the camera will move closer to the component.

- When our user **hover** his mouse over a component, the component will be lit up and there will be its name appearing on it.

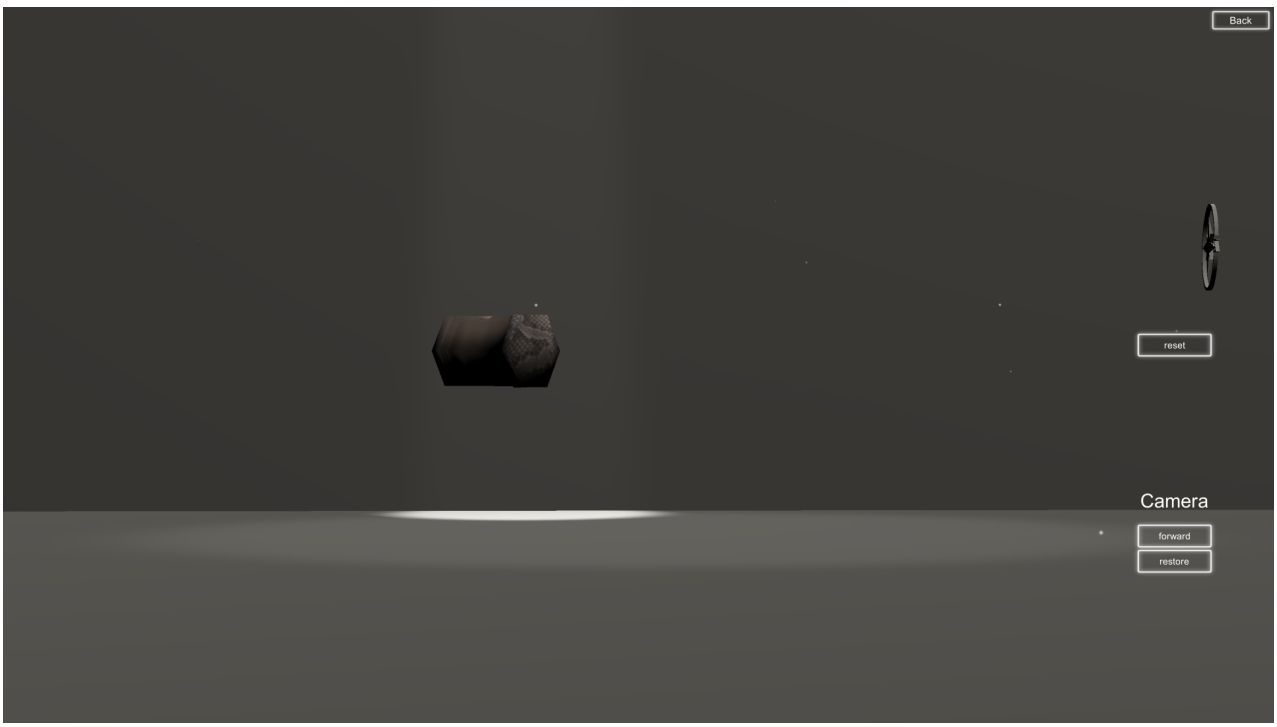
- When our user **double-clicks** on a component, he will enter the scene of the component.
- We provide a *Back* button for our user and he can **click** on it to return to the choose scene.

2.4 Interaction in componentScene



- We provide a *return* button for our user and when he **clicks** on it, he will return to the **instrumentScene**.
- We provide a *forward* button for our user and when he **clicks** on it, the camera will move forward and get more closer with the component.

- We provide a *return* button for our user and when he **clicks** on it, the camera will return to its origin place.
- We present a compass on the right, and when our user **clicks** on anywhere in the scene, the component will begin to rotate by a certain angle. We provide a *reset* button and when our user **clicks** on it, the component will return to its origin place.



3. Advantages and Disadvantages

3.1 Advantages

1. The biggest advantage of our project is that its interface and UI is fascinating. We apply some combined lights to generate special effect, which makes the scene seemed real.
2. The 3D Model of our project is lifelike.
3. The stucture of our program is concise and organized, so it is very easy for us to make some improvement.
4. Since this is a 3D display model interaction, we put the 3D model's presentation in a prominent position, show them with the most interface resources, cut off the extra functions and interface elements, so that the focus becomes more important, let The need for outstanding changes is more prominent.
5. By dividing the content displayed by the 3D model into different levels, the content diversity is increased, and the focus is more prominent.
6. At the same time, our main interaction logic is unified, allowing users to understand and master the functions that can be realized simply and clearly.

3.2 Disadvantages

1. We need to add more features to our project such as rotating by the mouse or basic introduction to the instruments.
2. There are still some bugs in our system.
3. We fail to port our program to our mobile phone.
4. We do not have much time after spending lots of time learning Unity3D, so we only apply the rotation to three components.
5. The function realization of user interaction needs to be improved. For example, there is no way to determine the zoomed position according to the position of the user's mouse when zooming, and the user cannot determine the direction and position of the rotation.
6. As far as the method of use is concerned, the guidance to the user is still insufficient. Users still need preliminary exploration to know the logic of interaction.
7. The style of each interface is not uniform enough. The position of the 3D model, the position of the button, and the setting of the environment such as light are still different in different interfaces, so that the user may have a sense of difference when jumping on different interfaces.

4. How to Improve Our Program

According to the analysis of our advantages and disadvantages, there are many things we can do to improve our program:

1. We can make full use of the Unity3D engine to add more features.
2. We can add some brief introductions of instruments and components.
3. It is convenient for us to port Unity3D to other platforms, so we could try this after we finish our projects.

5. Development Environment

- **Development Environment:**

  Win 10 & MacOS 10.14.5

- **Development Software:**

  1. **Unity3D** 2019.1.4.f1
  2. **Visual Studio 2017** *15.9.28307.665*

- **Development Language:**

C#

- **Others:**

  You can run the system on both operating systems.

6. Core Scripts

We have write many scripts which you can see in the *Script* folder of our project. But I only show some of the core scripts here.

nearComponents.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class nearComponent : MonoBehaviour
{
    public CameraMove camera;

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

        camera.targetPosition.x = transform.position.x;
        camera.targetPosition.y = transform.position.y+30;
        camera.targetPosition.z = transform.position.z-200;
    }
}
```

cameraBack.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class cameraBack : MonoBehaviour
{
    public CameraMove camera;
    private Vector3 originPosition;


    void Start()
    {
        originPosition = camera.transform.position;

        this.GetComponent<Button>().onClick.AddListener(delegate ()
        {
            camera.targetPosition = originPosition;
        });
    }
```

```
}
```

cameraForward.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class cameraForward : MonoBehaviour
{
    public CameraMove camera;
    public int distance;
    public int offsetX = 0;
    public int offsetY = 0;
    public GameObject obj;

    void Start()
    {
        this.GetComponent<Button>().onClick.AddListener(delegate ()
        {
            camera.targetPosition.x = obj.transform.position.x + offsetX;
            camera.targetPosition.y = obj.transform.position.y + offsetY;
            camera.targetPosition.z = obj.transform.position.z - distance;
        });
    }
}
```

CameraMove.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraMove : MonoBehaviour
    {
        public Transform target;
        public float smoothTime = 0.3F;
        private Vector3 velocity = Vector3.zero;
        public Vector3 targetPosition;
        public Vector3 originPosition;
        void Start()
        {
            targetPosition = transform.position;
            originPosition = transform.position;
        }
        void Update()
        {

            // Smoothly move the camera towards that target position
            transform.position = Vector3.SmoothDamp(transform.position, targetPosition, ref
velocity, smoothTime);
```

```
        }
    }
```

erhuController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class erhuController : MonoBehaviour
{
    private Vector3[] position = {
        new Vector3(-0.24f, -2.44f, 4.2f),      //前0
        new Vector3(-15.70f, -4.40f, 16.30f),      //左1
        new Vector3(-0.49f,-0.78f,24.89f),        //后2
        new Vector3(11.4f,-3.31f,13.57f),       //右3
    };
    private float smoothTime = 0.3F;
    private Vector3 velocity = Vector3.zero;

    private int now = 2;

    public Renderer rend;
    void Start()
    {
        rend = GetComponent<Renderer>();
        //rend.enabled = false;
    }

    void Update()
    {
        // Smoothly move the camera towards that target position
        transform.position = new Vector3(Mathf.SmoothDamp(transform.position.x,
position[(Controller.cnt + 2) % 4].x, ref velocity.x, smoothTime),
            Mathf.SmoothDamp(transform.position.y, position[(Controller.cnt + 2) % 4].y,
ref velocity.y, smoothTime),
            Mathf.SmoothDamp(transform.position.z, position[(Controller.cnt + 2) % 4].z,
ref velocity.z, smoothTime));

        now = (Controller.cnt + 2) % 4;
    }

    void OnMouseDown()
    {
        if (now == 1) { Controller.cnt = (4 + (Controller.cnt - 1)) % 4; }
        else if (now == 2)
        {
            Controller.cnt = (Controller.cnt + 1) % 4;
            Invoke("growth", 0.3f);
        }
        else if (now == 3) { Controller.cnt = (Controller.cnt + 1) % 4; }
```

```
    }

    void growth()
    {
        Controller.cnt = (Controller.cnt + 1) % 4;
    }
}
```

objRotateController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class objRotateController : MonoBehaviour
{
    public Quaternion targetRotation;
    private Quaternion originRotation;
    private float RotateAngle = 30;
    public float limit = 0.9f;

    void Start()
    {
        originRotation = Quaternion.Euler(transform.eulerAngles) * Quaternion.identity;
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            if (transform.rotation.y < limit)
            {
                Rotate();
            }
        }
        if (Input.GetMouseButton(0))
        {
            if (transform.rotation.y < limit)
            {
                Rotate();
            }
        }
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation,
Time.deltaTime * 3);//利用Slerp插值让物体进行旋转  2是旋转速度 越大旋转越快
    }

    public void Rotate()
    {
        float x = transform.eulerAngles.x;
        float y = transform.eulerAngles.y + RotateAngle;
        float z = transform.eulerAngles.z;
```

```
        targetRotation = Quaternion.Euler(new Vector3(x, y, z)) * Quaternion.identity;  //给旋转目标值赋值，由于只有Y轴动，所以目标值应是  (旋转角(RotateAngle)*需要旋转的个数(count)+origionY(物体初始Y轴旋转角))*Quarternion.identity
    }
}
```

rotateController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class rotateController : MonoBehaviour
{
    public objRotateController obj;
    private Quaternion compass_targetRotation;

    void Start()
    {
        compass_targetRotation = Quaternion.Euler(new Vector3(91.5f, 88.5f, -82.5f)) * Quaternion.identity;
    }

    void Update()
    {
        compass_targetRotation = Quaternion.Euler(new Vector3(91.5f, 88.5f, -82.5f + obj.targetRotation.y * 180.0f)) * Quaternion.identity;

        transform.rotation = Quaternion.Slerp(transform.rotation, compass_targetRotation, Time.deltaTime * 3);//利用Slerp插值让物体进行旋转  2是旋转速度 越大旋转越快

    }


}
```

LevelChange.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LevelChange : MonoBehaviour
{
    public Animator animator;
    public startGame startGame;
```

```
    public int levelToLoad;
    // Update is called once per frame
    void Update()
    {
        if(startGame.bStart)
        {
            fadeToLevel(levelToLoad);
        }
    }

    public void fadeToLevel(int levelIndex)
    {
        animator.SetTrigger("bFadeOut");
    }

    public void fadeOutComplete()
    {
        SceneManager.LoadScene(levelToLoad);
    }

}
```

LevelChangeForModels.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;


//绑定LevelChangeFoeModel之后绑定相应model的ClckParts的script

public class LevelChangeForModels : MonoBehaviour
{
    public Animator animator;
    public clickParts clickParts;

    private int levelToLoad;

    private void Start()
    {
        levelToLoad = clickParts.levelToMove;
    }

    // Update is called once per frame
    void Update()
    {
        if (clickParts.startToChange)
        {
            fadeToLevel(levelToLoad);
        }
```

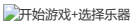
```
    }

    public void fadeToLevel(int levelIndex)
    {
        animator.SetTrigger("bFadeOut");
    }

    public void fadeOutComplete()
    {
        SceneManager.LoadScene(levelToLoad);
    }
}
```

7.Video for preview

Start & Choose



Instrument Zither

Instrument Lute

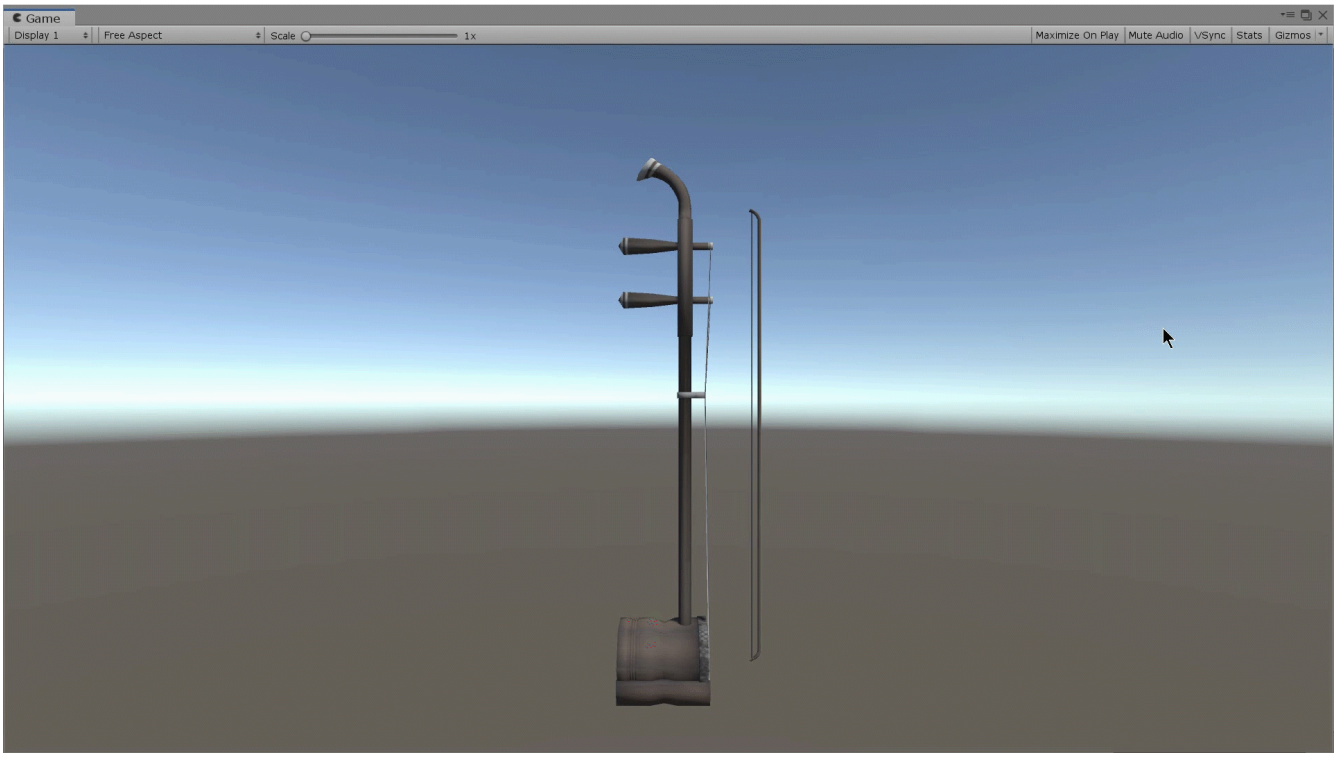
Instrument Erhu

Decomposition of Lute

Decomposition of Erhu

Rotation for Erhu

It haven't added into the final project because of the immature.



8.About the author

ID	Name
1754060	Zhe Zhang(张喆)
1751894	Le Yang(杨乐)
1753188	Kaixin Chen(陈开昕)

adviser Ying Shen

contact email: doubleZ0108@gmail.com