项目说明文档

竞教结合课程项目——京韵

——一款融合京剧元素的新概念音游

作　　者: 黄彦铭 2050865　刘一飞 2051196

指 导 教 师: 朱宏明

学院、 专业: 软件学院 软件工程

同济大学

Tongji University

目录

一、 项目综述

此项目为一款结合了京剧国粹的音乐节奏游戏，游戏内集成了多种游玩模式，涵盖了射击元素、跑酷元素、对战元素，让用户在体验不同类型的音乐节奏游戏的同时，感受到京剧的魅力。

二、项目采用的工具

项目采用 unity 进行开发，使用 C#语言给游戏对象挂载脚本。

三、应用场景

此项目的目标用户为热爱玩游戏或需要游戏调剂生活的青少年群体，以及热爱京剧或对京剧有一定兴趣的群体。用户能够在游玩的过程中收获丰富的游戏体验，并且对京剧产生更加浓厚的兴趣。

四、解决的问题

此项目旨在解决传统国粹——京剧在青年群体中认知度较低的问题，将京剧以一种顺应时代发展的方式呈现出来，焕发新的活力，为青年群体提供一个了解京剧的平台，并通过游戏的方式提升他们对京剧兴趣，最终提升整个青年群体对于京剧这一传统国粹的认知度。

此项目解决的另一个问题，是音游玩法的局限性问题。当前市面上的绝大多数音游都是 2D 界面，缺乏在高维度上的探索。本项目创造性地打破了这种传统约束，将 3D 特有的射击与跑酷元素结合进音游，实现音游的创新。

与此同时，此项目也为热爱京剧的人群提供了更加新颖的娱乐方式，丰富了人们的日常生活。

五、项目界面与功能

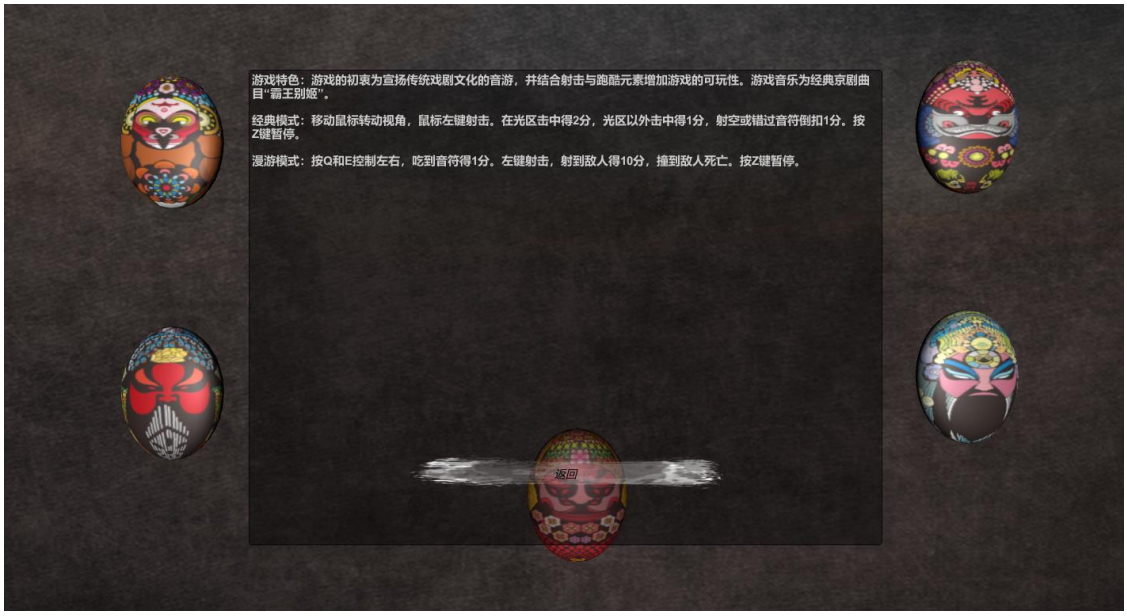
本项目有一个起始页面，两个游戏模式，以下对每个页面的功能进行详述。

1. 主菜单

主菜单有三个按钮，开始、说明、退出。

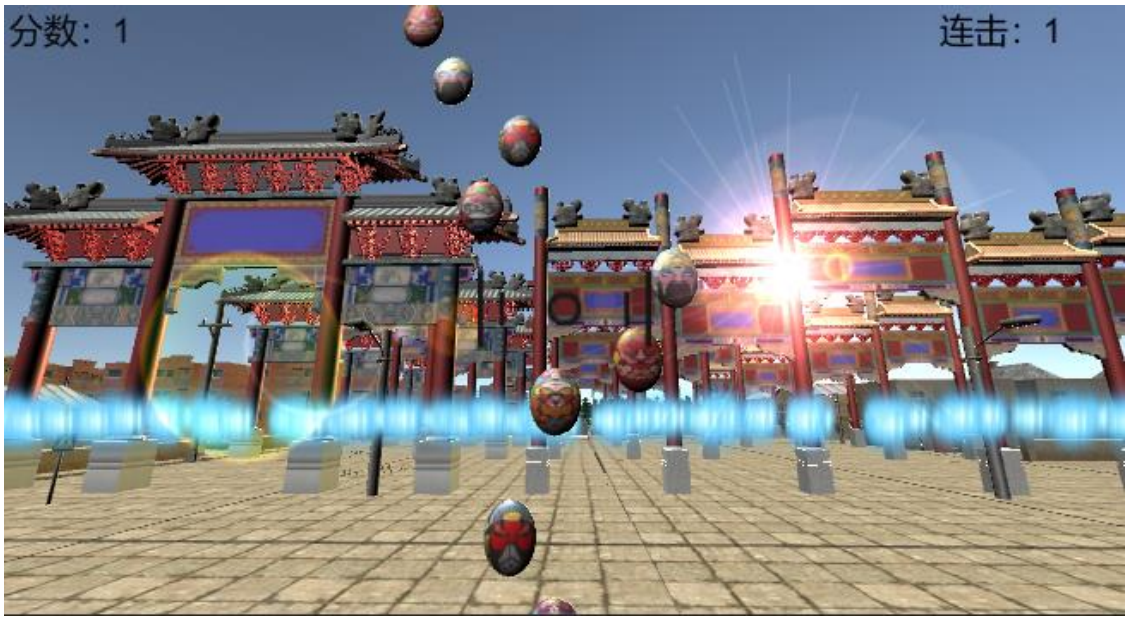
点击“开始”按钮会跳转到选择模式的菜单。“经典模式”和“漫游模式”两个按钮可以进入对应的游戏模式，“返回”按钮可以跳转回上一级菜单。

点击“说明”按钮会跳转到说明页面，展示游戏简介与玩法说明。点击“返回”按钮可以跳转回上一级菜单。



2. 经典模式——射击类音游

在经典模式中，游戏场景为古风建筑。在面前不断有脸谱样式的音符下落。点击鼠标左键可以发射子弹击打音符，音符也会放出声音。若在蓝光区域打中加 2 分，再其他区域打中加 1 分，击空或错过音符均扣除 1 分。左上角会显示当前分数，右上角会显示当前的连击数。



游戏支持暂停功能，按 Z 键可以暂停。在暂停界面可以选择继续游戏或是返回主菜单。

当所有音符全部下落后，来到结束页面，并显示最终得分和最大连击数。

3. 漫游模式——跑酷类音游

在漫游模式中，场景为小镇街道。人物自动向前跑，按 Q 和 E 控制左右。吃到音符会演奏音符，并获得一分。左上角显示当前分数。

漫游模式页支持按 Z 键暂停，并显示当前分数。可以在暂停框选择继续游戏或是返回主菜单。

点击鼠标左键可以发射箭矢，射击地图中的敌人，射中敌人加 10 分。

如果碰到敌人，则敌人播放砍人动画，主角播放倒地动画。

当主角被敌人打倒或是到达终点时，游戏结束，并显示最终得分。

六、项目代码示例

1.乐谱的存储与读取

游戏的音符按照乐曲的音符与节奏进行，所以需要在外部文件中存储相应的乐谱信息。采用 XML 格式在外部存储乐谱，存储内容有音符的音高和节奏，需要具备一些基础的乐理知识。

在开始游戏时，乐谱由外部调入内存，将音高和节奏存入内存中的 NoteType 和 TimeType 数组中，后续经过代码解析生成游戏场景中的脸谱音符。

```
public void LoadMusic()//装载乐谱
{
    int i = 1;
    XmlDocument xml = new XmlDocument();
    xml.Load(Application.streamingAssetsPath + FileName);
    XmlNodeList list = xml.SelectSingleNode("Save").ChildNodes;
    foreach (XmlElement xe in list)
    {
        NoteType[i] = xe.GetAttribute("pitch").ToString();
        TimeType[i] = xe.GetAttribute("zone").ToString();
        i++;
    }
}
```

```
<Save>
  <A pitch="E4" zone="four" />
  <A pitch="E4" zone="four" />
  <A pitch="E4" zone="four" />
  <A pitch="E4" zone="four" />
  <A pitch="D4" zone="four_" />
  <A pitch="E4" zone="eight" />
  <A pitch="G4" zone="four" />
  <A pitch="A4" zone="four" />
```

2.经典模式中音符下落节奏的控制

音符的节奏，在程序中体现为一个音符距离下一个音符出现的时间间隔，需要调整相应的延时达到按节奏下落的效果。

编写一个计时器函数 Timer。在 Timer 中可以明确现在正在计算哪两个音符之间的时间间隔。时间的增长通过调用 Time 类中的 deltaTime 成员，可以实现计时效果。

```
private void Timer(float waitTime, int NumTimer)
{
    if (TimerOn[NumTimer])
    {
        timer += Time.deltaTime;
        if (timer >= waitTime)
        {
            timer = 0;
            TimerOn[NumTimer] = false;//计时器关闭
            NoteOn[NumTimer + 1] = true;//开启下一个音符
            ite++;
        }
    }

}
```

音符的创建需要在游戏中生成相应的对象，通过调用 GameObject 类中的 Instantiate 来实现。生成的音符在游戏中有相应的预制体，根据每个音符的音高选择相应的预制体进行生成即可。

```
private void NewNote()
{
    GameObject.Instantiate(note, NotePosition.transform.position, NotePosition.transform.rotation);
}
```

将以上代码放在 Unity 中的 Update 函数中调用。Update 函数每一帧都会调用一次，以此进行反复地触发检测，直到整个乐谱被演奏完毕。

3.音符的消除与演奏

在两个模式中，音符都是因发生碰撞而消除并播放音效的。要实现这个功能，首先要在 Unity 中为音符的实体添加触发器组件与刚体组件，并在物体挂载的 C#脚本中的 OnTriggerEnter()函数写入碰撞时相应的逻辑。

从以下代码中可以看到，在发生碰撞时，通过 AudioSource 类中的 PlayClipAtPoint()方法播放相应音高的音频，然后关闭触发器以避免二次击中而加分，最后利用 Destroy()函数消除游戏中的实体。

```
void OnTriggerEnter(Collider collider)
{
    IsDestroyed = 1;
    MusicDeath();
}
2 个引用
private void MusicDeath()
{
    AudioSource.PlayClipAtPoint(audioClip, transform.localPosition);
    gameObject.GetComponent<SphereCollider>().enabled = false;//关闭触发器，防止在播放死亡动画时造成二次击中加分
    Destroy(gameObject);
}
```

4. 人物动作的实现

人物的模型与动作使用的是现成资源，但需要通过一定改造后加入自己的程序。在人物模型资源本身的设定中，按下 W 键可以使人物向前奔跑，所以在游戏中为了实现持续向前奔跑，采用虚拟按键来始终向程序中传入 W 键被按下的外设信息。

```
//虚拟按键实现持续前进
[DllImport("user32.dll", EntryPoint = "keybd_event")]
8 个引用
public static extern void keybd_event(
byte bVk,     //虚拟键值 对应按键的ascll码十进制值
byte bScan,// 0
int dwFlags,  //0 为按下，1按住，2为释放
int dwExtraInfo  // 0
);
```

而当人物死亡后，应停止前进，播放死亡动画。此时需要发送释放键盘的信息，并调用人物动作的 API 播放死亡动画。

```
if (is_death)
{
    //撤销虚拟按键
    keybd_event(87, 0, 2, 0);//按W，W的ASCII码是87
    keybd_event(83, 0, 2, 0);//按S，S的ASCII码是83
    keybd_event(68, 0, 2, 0);//按D，D的ASCII码是68
    keybd_event(65, 0, 2, 0);//按A，A的ASCII码是65
    //播放死亡动画
    animator.SetTrigger("Death");
}
```

5. 相机跟随人物移动的实现

相机跟随人物移动，需要考虑到两点。第一，相机与人物之间的相对位置需要保持不变，即相机与人物之间构成的三维向量需要保持恒定；第二，相机的视角需要随着人物转向而旋转。

在刚进入游戏时，就需要记录下相机与人物之间构成的三维向量，之后需要保证二者的相对为止不发生变化，代码如下：

```
void Start()
{
    MoveMode = CharacterControllerSimple.MoveMode;
    initialDistanceY = transform.position.y - target.position.y;
    initialDistanceXZ = transform.position.x - target.position.x;
}
```

由于在游戏场景中，人物只能朝四个方向走，所以为人物移动设置四个模式 MoveMode：X+,X-,Z+,Z-，代表人物正在朝哪个坐标轴移动。

根据人物当前的 MoveMode，决定人物向哪个方向移动，代码如下：

```
if(MoveMode=="Z+")
{
    transform.position = Vector3.Lerp(transform.position, new Vector3(target.transform.position.x,
                                      target.transform.position.y + initialDistanceY,
                                     target.transform.position.z + initialDistanceXZ), Time.deltaTime * 10);
}
else if(MoveMode=="Z-")
{
    transform.position = Vector3.Lerp(transform.position, new Vector3(target.transform.position.x,
                                      target.transform.position.y + initialDistanceY,
                                     target.transform.position.z - initialDistanceXZ), Time.deltaTime * 10);
}
else if (MoveMode == "X+")
{
    transform.position = Vector3.Lerp(transform.position, new Vector3(target.transform.position.x + initialDistanceXZ,
                                      target.transform.position.y + initialDistanceY,
                                     target.transform.position.z), Time.deltaTime * 10);
}
else if (MoveMode == "X-")
{
    transform.position = Vector3.Lerp(transform.position, new Vector3(target.transform.position.x - initialDistanceXZ,
                                      target.transform.position.y + initialDistanceY,
                                     target.transform.position.z), Time.deltaTime * 10);
}
```

根据人物相邻两次的 MoveMode，可以确定人物的转向，是左转还是右转，代码如下：

```
string previous = MoveMode;
string current = CharacterControllerSimple.MoveMode;
//判断左转还是右转
if (previous == "Z+" && current == "X-" ||
    previous == "X-" && current == "Z-" ||
    previous == "Z-" && current == "X+" ||
    previous == "X+" && current == "Z+")
    TurnDirection = "left";
else if (previous == "Z+" && current == "X+" ||
    previous == "X+" && current == "Z-" ||
    previous == "Z-" && current == "X-" ||
    previous == "X-" && current == "Z+")
    TurnDirection = "right";
```

而根据人物的转向情况，便可以对摄像机的 rotation 参数进行更改。在更改时调用 transform 对象的 RotateAround()方法，让其值进行逐渐地变化，以体现相机转向的动态视觉效果。旋转过程放在 Update 函数中，Update 函数每一帧都会调用，可以实现持续地旋转。当旋转的角度达到要求时，便停止旋转。

```
if (TurnDirection == "left")
{
    this.transform.RotateAround(target.position, Vector3.up, -Time.deltaTime * RotateSpeed);
    SumRotate = 90f;
}
else if (TurnDirection == "right")
{
    this.transform.RotateAround(target.position, Vector3.up, Time.deltaTime * RotateSpeed);
    SumRotate = 90f;
}

CurrentRotate += Time.deltaTime * RotateSpeed;
if (SumRotate - CurrentRotate < 1e-2)//转过的角度达到要求，则停止旋转
{
    IsRotating = false;

}
```

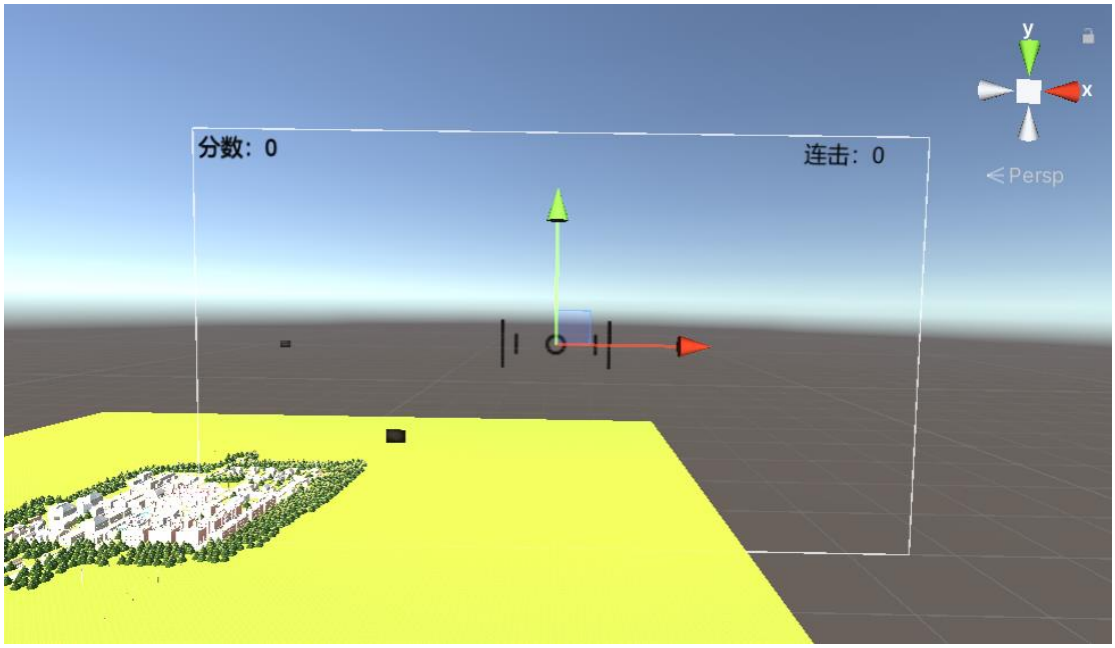
6. UI 界面的实现

项目中的 UI 界面使用到了两种方法：一种是在 C#脚本中的 OnGUI 函数中进行编写，此方法较为传统，方便页面跳转；另一种方法是利用 unity 中提供的 canvas 画布，在画布上进行可视化的组件拖拽，实现可视化的 UI 界面设计。

在 C#脚本中编写 UI 界面的方法，以开始界面为例。通过定义一个字符串 menuPage 记录当前在哪个页面并在 if 语句中写入 GUI 类中的方法，来显示 UI 组件。在定义 UI 组件时需要利用 Rect 类来定义一个四元组，来给出组件的大小。

```
//开始菜单
else if (menuPage == "开始")
{
    if(GUI.Button(new Rect(playButton),"经典模式"))
    {
        //audio.PlayOneShot(beep);
        Classical_Human();//脚本与物体启用
        Cursor.visible = false;
        StartCoroutine("ButtonAction", "scene");
    }
    if(GUI.Button(new Rect(instructionsButton),"漫游模式"))
    {
        //audio.PlayOneShot(beep);
        Create.ModeTwoStart = true;//开始存人物信息
        RunMode();
        StartCoroutine("ButtonAction", "Demo_Scene");
    }
    if (GUI.Button(new Rect(quitButton), "返回"))
    {
        //audio.PlayOneShot(beep);
        menuPage = "main";
    }
}
```

另一种方法是在 unity 中采用可视化方法拖拽组件，主要用于游戏场景中的 UI 设计。通过拖拽组件实现 UI 组件的样式和位置，并挂载需要的 C#脚本实现 UI 组件的动态变化，例如分数的动态变化。


分数：0
连击：0
y
x
Persp