设计思路

本项目设计思路围绕创建一个“趣味桌球游戏”，主要使用游戏对象（Ball）等多种原理，利用 unity C#的开发环境来完成游戏的实现。在本项目的实现过程中，主要通过以下思路和步骤完成游戏的制作。

系统完整功能描述及关键代码展示

1.1 场景布置

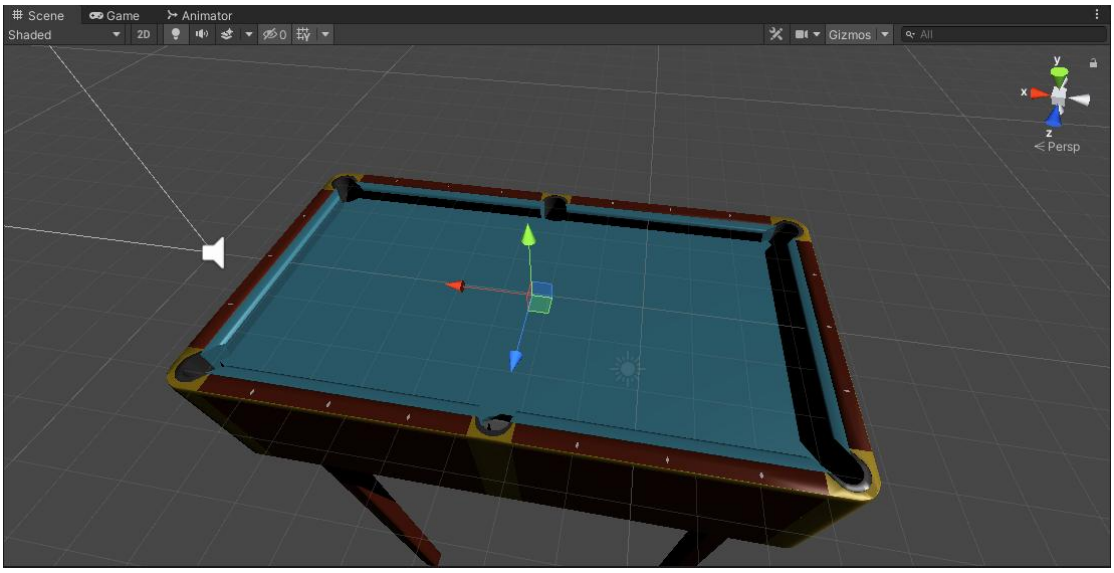
在游戏中主要有开始场景，游戏场景以及结算场景，其中开始场景，胜利场景和失败场景都是由 UI 搭建实现，在这三个场景中，都采用 Image 来作为页面的背景，然后采用 Text 用于页面中的文字显示，例如游戏标题，胜利失败文字显示，最后采用 Button 来作为页面中的按钮，这里实现的页面效果图如下：

1.2 场景切换关键代码

在开始页面中，有开始游戏和退出游戏两个按钮事件，分别使用加载场景和退出游戏的函数来实现，在结算页面中，首先需要根据当前游戏结果显示对应的胜利还是失败页面，在两个页面中，需要使用切换场景的代码来实现再来一次和下一关的按钮事件，这里的主要代码如下：

```
public class Bag : MonoBehaviour
{
    public void Tz(int i)
    {
        SceneManager.LoadScene(i);
    }

    public void Exit()
    {
        Application.Quit();
    }
}
```

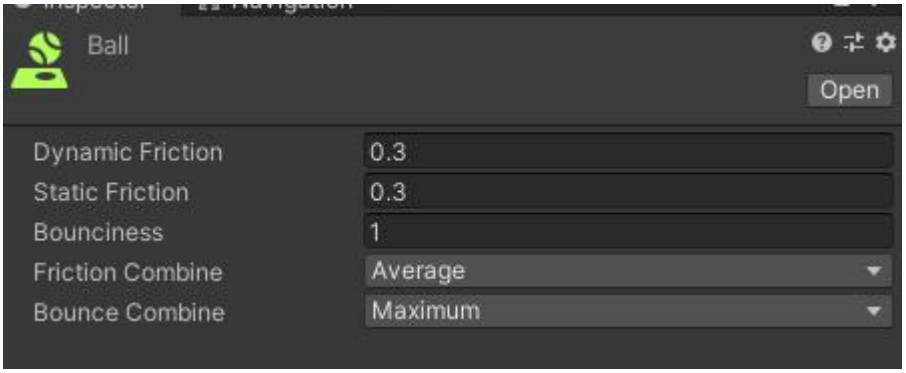
游戏场景为 3D 场景，此场景由台球桌来作为整个游戏的场景，这里需要采用 3Dmax 来制作一个台球桌模型，此处的台球搭建实现，首先需要使用 Terrain 组件的设置，首先收集参考图片或资料，确定模型的大小和比例。使用 3Dmax 的基础模型工具，制作一个基础模型。制作基础模型的方式可以使用盒子模型，根据参考图像划分每个组件并手动制作组件。使用 3Dmax 的材质编辑器，创建表面的纹理和材质，使其看起来更逼真。 添加台球桌的细节。在 3Dmax 中制作台球桌的门口和球洞，并设定封闭的物件，确保球经过时进入球洞。最后把实现的台球桌模型导出 FBX 并导入到 unity 中，效果图如下：



1.3 主球控制实现

游戏中的主球首先需要给其加入刚体和碰撞体，这样主球就会具有重力和碰撞等物理效果，不过此处还需要主球具有摩擦力，弹力等效果，所以需要给主球

加入一个物理材质，在 Unity 中，物理材质可以影响游戏对象的摩擦力、弹性、摆动等物理属性。提供了各种物理属性选择项和调整工具，能够自由调整物体的摩擦力、弹性等属性。这里设置好的主角物理材质如下：

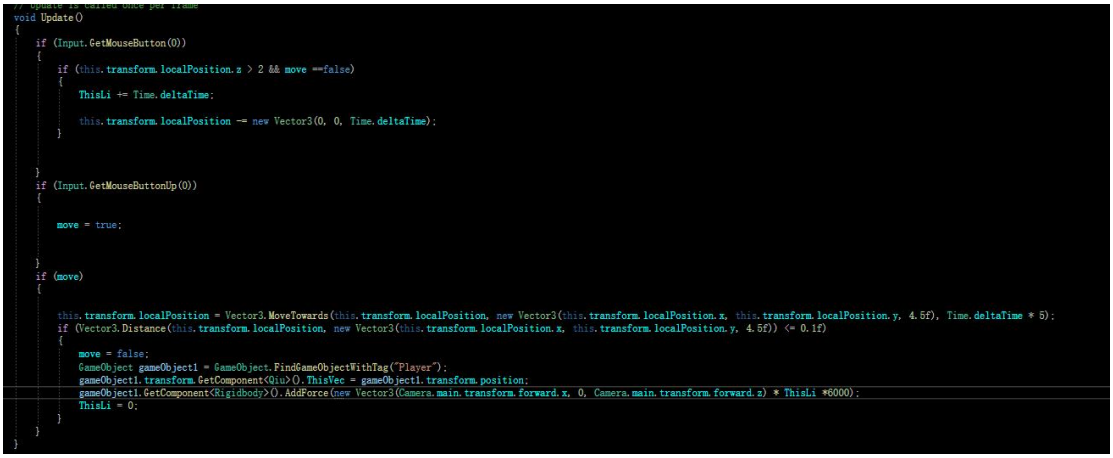


1.4 主球移动关键代码

在主球的子物体下创建一个摄像机，通过调整摄像机的位置和视角旋转使得摄像机处于主球后上方看向主球，这里的摄像机需要可以进行左右移动，这里通过 Input 的方法获取玩家的按钮按键（a 和 d）状态，当玩家按下对应的按键后，对摄像机进行左右旋转，当主球移动时，还需要控制摄像机移动向主球，来实现摄像机跟随，这里的主要代码如下：

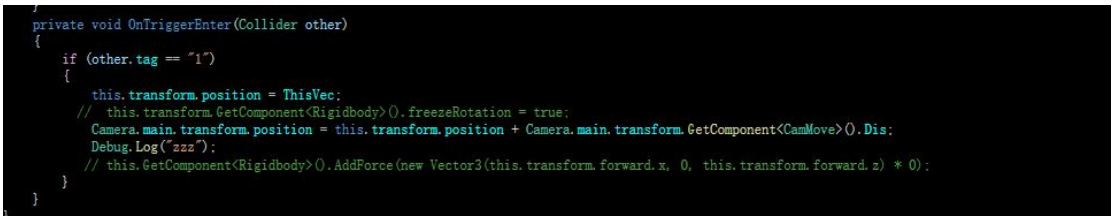
```
// Update is called once per frame
void Update()
{
    if (Player.transform.GetComponent<Rigidbody>().velocity.x == 0 && Player.transform.GetComponent<Rigidbody>().velocity.y == 0 && Player.transform.GetComponent<Rigidbody>().velocity.z == 0)
    {
        this.transform.GetChild(0).gameObject.SetActive(true);
        if (Input.GetKey(KeyCode.A))
        {
            this.transform.RotateAround(Player.transform.position, Vector3.up, Time.deltaTime *40);
            Dis = this.transform.position - Player.transform.position;
        }
        if (Input.GetKey(KeyCode.D))
        {
            this.transform.RotateAround(Player.transform.position, -Vector3.up, Time.deltaTime * 40);
            Dis = this.transform.position - Player.transform.position;
        }
    }
    else
    {
        this.transform.GetChild(0).gameObject.SetActive(false);
        this.transform.position = Vector3.Lerp(this.transform.position, Player.transform.position + Dis, Time.deltaTime * 3);
    }
}
```

在摄像机子物体下创建一个球杆，此处球杆身上加入代码，代码中检测鼠标左键的按下状态，当持续按下时，让球杆向后移动，这里判断移动的距离，当移动到最远距离时，停止移动，并且蓄力也不再增加，当抬起鼠标左键时，控制球杆迅速向前移动一段距离，移动结束后，根据摄像机当前的朝向，给与主球一个此方向的力，同时隐藏球杆，这里的主要代码如下：



```
// Update is called once per frame
void Update()
{
    if (Input.GetMouseButton(0))
    {
        if (this.transform.localPosition.z > 2 && move ==false)
        {
            ThisLi += Time.deltaTime;

            this.transform.localPosition -= new Vector3(0, 0, Time.deltaTime);
        }


    }
    if (Input.GetMouseButtonUp(0))
    {

        move = true;


    }
    if (move)
    {

        this.transform.localPosition = Vector3.MoveTowards(this.transform.localPosition, new Vector3(this.transform.localPosition.x, this.transform.localPosition.y, 4.5f), Time.deltaTime * 5);
        if (Vector3.Distance(this.transform.localPosition, new Vector3(this.transform.localPosition.x, this.transform.localPosition.y, 4.5f)) <= 0.1f)
        {
            move = false;
            GameObject gameObject1 = GameObject.FindGameObjectWithTag("Player");
            gameObject1.transform.GetComponent<Qiu>().ThisVec = gameObject1.transform.position;
            gameObject1.GetComponent<Rigidbody>().AddForce(new Vector3(Camera.main.transform.forward.x, 0, Camera.main.transform.forward.z) * ThisLi *6000);
            ThisLi = 0;
        }
    }
}
```

最后在球被击打时，需要记录当前位置，然后对球加入碰撞检测代码，当主球进洞时，让主球以及摄像机的坐标都改变为击打时记录的坐标，实现归位的操作，这里的主要代码如下：



```
}
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "1")
    {
        this.transform.position = ThisVec;
      //  this.transform.GetComponent<Rigidbody>().freezeRotation = true;
        Camera.main.transform.position = this.transform.position + Camera.main.transform.GetComponent<CamMove>().Dis;
        Debug.Log("zzz");
       // this.GetComponent<Rigidbody>().AddForce(new Vector3(this.transform.forward.x, 0, this.transform.forward.z) * 0);
    }
}
```

1.5 其他功能实现

倒计时：游戏中加入 Text 来显示倒计时，并创建一个值来记录当前倒计时，并在代码中一直减少倒计时的值，如果倒计时完毕，就会根据当前进洞的球个数显示对应的成功失败界面，这里的主要代码如下：

```
{
//  Debug.Log(Score);
  Time1 -= Time.deltaTime;

  if (i == false)
  {
      TimeText.text = "剩余时间：" + (int)Time1;
      if (Time1 <= 0)
      {
          StateQIU.SetActive(false);
          leveQIU.SetActive(true);
          TimeText.gameObject.SetActive(false);
      }
  }
  if (i == true)
  {
      TimeText.text = "剩余时间：" + (int)Time2;
      Time2 -= Time.deltaTime;
      if (Time2 <= 0)
      {
          Shib.SetActive(true);
      }
  }
```

得分：在其他小球身上加入碰撞检测代码，当检测到自身进洞后，给与玩家加分反馈，这里的主要代码如下：

```
}
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "1")
    {
        Destroy(this.gameObject);
        GameObject.Find("Canvas").GetComponent<Gamemanager>().Score+=1;
    }
}
```

天空盒：游戏中的天空盒通过摄像机添加，天空盒是一种用于渲染游戏中场景背景的方法。它是一种高度优化过的技术，可以在不占用过多计算资源的情况下为游戏场景创建逼真的天空背景。 自定义天空盒可以基于 2D 或 3D 素材进行创建，并且可以调整天空盒的各种属性。 在使用天空盒的过程中，可以为天空盒选择不同的图片和材质，并且可以调整天空盒的亮度、材质颜色、太阳位置等属性，以便为不同的场景创造不同的氛围和环境。

分工说明

代码开发：

场景布置及场景切换：周翰栋

主球控制移动及摄影机和球杆的移动：毛舒琦 叶梦飞

其他补充功能（倒计时、得分、天空盒）：汤洒、杜俊贤

文档编写：毛舒琦 叶梦飞

PPT 制作：杜俊贤

PPT 讲解：周翰栋