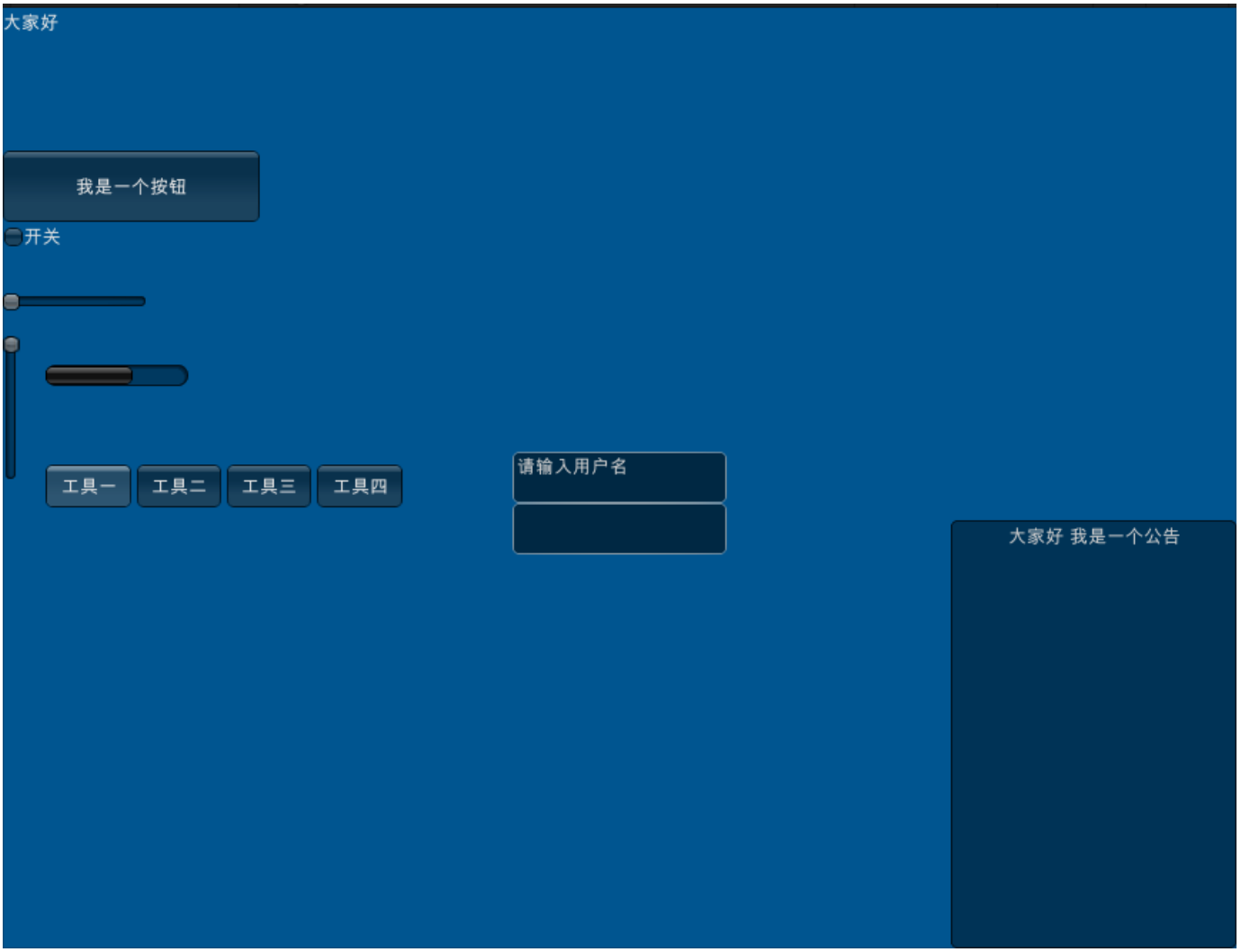
OnGUI系统

最老的UI系统 性能消耗比较大 开发起来比较麻烦

逐步被淘汰 但快速帮我们DEBUG和添加功能




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

public class UI : MonoBehaviour {

    Rect rect = new Rect(0, 0, 500, 500);

    bool toggle;//开关的真假值
    float hValue;//水平拖动条的float值
    float vValue;//垂直拖动条的float值
    float vScrollValue;//水平卷动条的float值

```

```
    int selected;//工具选项

    string username = "请输入用户名";
    string password ="";

    //所有跟UI相关的逻辑 都必须放在OnGUI里执行
    void OnGUI()
    {
        //文本标签
        GUI.Label(new Rect(0, 0, 100, 100), "大家好");

        //按钮
        if (GUI.Button(new Rect(0, 100, 180, 50), "我是一个按钮"))
        {
            Debug.Log("我被按下了");
        }

        //开关 应用于是否开启功能
        toggle=GUI.Toggle(new Rect(0, 150, 100, 50), toggle, "开关");

        //水平拖动条
        hValue=GUI.HorizontalSlider(new Rect(0, 200, 100, 30), hValue,
0, 100);

        //垂直拖动条
        vValue=GUI.VerticalSlider(new Rect(0, 230, 30, 100), vValue,
0, 1);
        //水平卷动条
        vScrollValue=GUI.HorizontalScrollbar(new Rect(30, 250, 100,
300), vScrollValue, 5.0f, 1, 10);

        //工具栏  互斥的选择项
        selected=GUI.Toolbar(new Rect(30, 320, 250, 30), selected, new
string[] { "工具一", "工具二", "工具三", "工具四" });

        switch (selected)
        {
            case 0:
                Debug.Log("选中了第一个");
                break;
            case 1:
                Debug.Log("选中了第二个");
```

```
                break;
            case 2:
                Debug.Log("选中了第三个");
                break;
            case 3:
                Debug.Log("选中了第四个");
                break;
        }

        //普通文本输入框
        username=GUI.TextField(new Rect(Screen.width / 2 - 75, Screen.height / 2 - 18, 150, 36), username);

        //密码输入框 char是掩码字符
        password = GUI.PasswordField(new Rect(Screen.width / 2 - 75, Screen.height / 2 + 18, 150, 36),
                password,'@',8);

        GUI.Box(new Rect(Screen.width - 200, Screen.height - 300, 200, 300), "大家好 我是一个公告");
    }
}

```

图片跟随鼠标位置移动

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

public class UI : MonoBehaviour {

    public Texture texture;

    private Rect rect;
    private float x;
    private float y;

    void Update()
```

```
    {
        if (Input.GetMouseButton(0))
        {
            x = Input.mousePosition.x;
            //把屏幕坐标的Y值转化为UI坐标的Y值
            y = Screen.height - Input.mousePosition.y;
        }
    }
    void OnGUI()
    {
        rect = new Rect(x-texture.width/2, y-texture.height/2,
texture.width, texture.height);
        GUI.DrawTexture(rect, texture);
    }
}
```

使用GUILayout自动布局

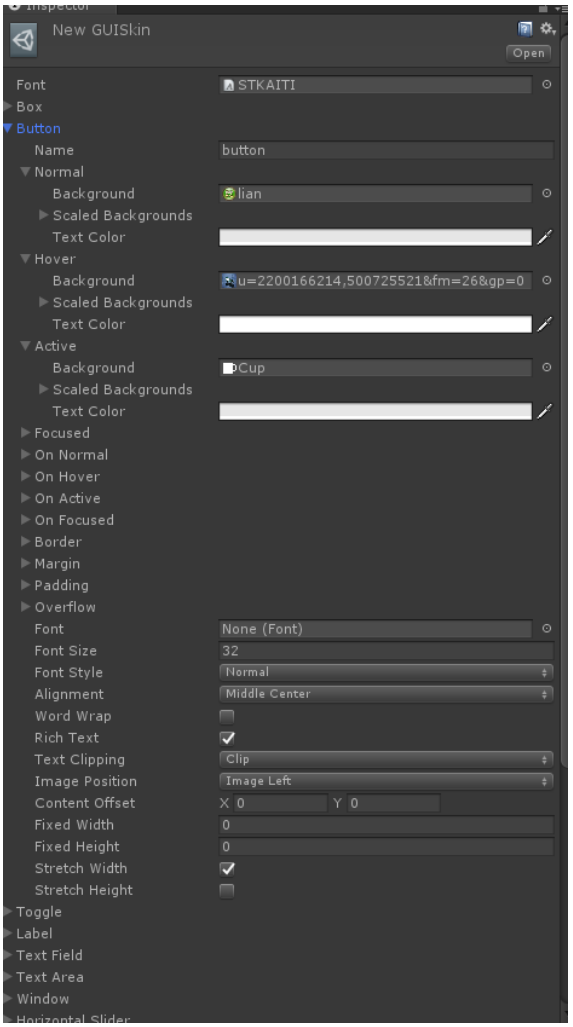
```
  void OnGUI()
    {
        if (GUILayout.Button("按钮1"))
        {

        }
        if (GUILayout.Button("按钮2"))
        {

        }
        //自动排版 并且设置宽高
        if (GUILayout.Button("按钮3", GUILayout.Width(150),
GUILayout.Height(50)))
        {

        }
        //自动排版 并且设置宽高
        if (GUILayout.Button("按钮4", GUILayout.Width(150),
GUILayout.Height(50)))
        {

        }
```

```
    }
```

GUI Skin皮肤

```
public class UI : MonoBehaviour {

    public Font font;
    public Font font2;

    void OnGUI()
    {
        //更换字体为楷体
        GUI.skin.font = font;

        //设置文本标签字体大小为48
        GUI.skin.label.fontSize = 48;

        //更改颜色
        GUI.color = Color.yellow;

        //更改背景色
        GUI.backgroundColor = Color.red;

        GUILayout.Label("48号楷体字");

        //设置按钮字体大小为32
        GUI.skin.button.fontSize = 32;
        GUILayout.Button("黄色按钮",GUILayout.Width(300));

        GUI.skin.font = font2;
        GUILayout.Label("黑体字");
    }
}
```

再Assts目录中创建GUISkin



Button的几种状态

1 Normal 正常显示效果

2 Hover 当鼠标悬浮时的效果

3 Active 当按钮被鼠标按下时的效果

```
public class UI : MonoBehaviour {
    //把外部创建的GUISkin 拖拽赋值
    public GUISkin gs;
    void OnGUI()
    {
        //把UI皮肤设置为自己创建的皮肤
        GUI.skin = gs;
        if (GUI.Button(new Rect(0f,0f,100f,100f), ""))
        {
            Debug.Log("按下");
        }
    }
}
```

Unity场景切换

```
    void OnGUI()
    {
        if (GUILayout.Button("切换到场景2"))
        {
            //老版本切换场景方法 只能通过传递场景序号来实现
            //Application.LoadLevel("Scene2");
            Application.LoadLevel(1);
        }
    }
```

5.X的场景管理和场景类型

```
public struct Scene
{

    public static bool operator !=(Scene lhs, Scene rhs);
    public static bool operator ==(Scene lhs, Scene rhs);

    public int buildIndex { get; }
    public bool isDirty { get; }
    public bool isLoaded { get; }
    public string name { get; internal set; }
    public string path { get; }
    public int rootCount { get; }

    public override bool Equals(object other);
    public override int GetHashCode();
    public GameObject[] GetRootGameObjects();
    public void GetRootGameObjects(List<GameObject> rootGameObjects);
    public bool IsValid();
}
```

使用SceneManager进行场景切换

```
public class UI : MonoBehaviour {

    void Start()
    {
```

```
        //创建新场景并加载到当前场景中
        SceneManager.CreateScene("Scene3");

        //获取当前正在运行的场景
        Scene s1= SceneManager.GetActiveScene();
        Debug.Log(s1.buildIndex);//场景索引
        Debug.Log(s1.path);//场景路径
        Debug.Log(s1.name);//场景名字

        SceneManager.GetSceneByBuildIndex(1);//通过索引获取场景
        SceneManager.GetSceneByName("Scene2");//通过场景名获取场景
        SceneManager.GetSceneByName("Assets/Scene2.unity");//通过场景路径获取场景
    }


    void OnGUI()
    {
        if (GUILayout.Button("切换到场景2"))
        {
            //通过场景名切换场景 默认模式是Single
            //SceneManager.LoadScene("Scene2");
            //SceneManager.LoadScene(1); 通过索引值切换场景

            //Additive模式 实现场景的叠加
            SceneManager.LoadScene("Scene2", LoadSceneMode.Additive);
        }
    }
}
```

场景切换的流程：

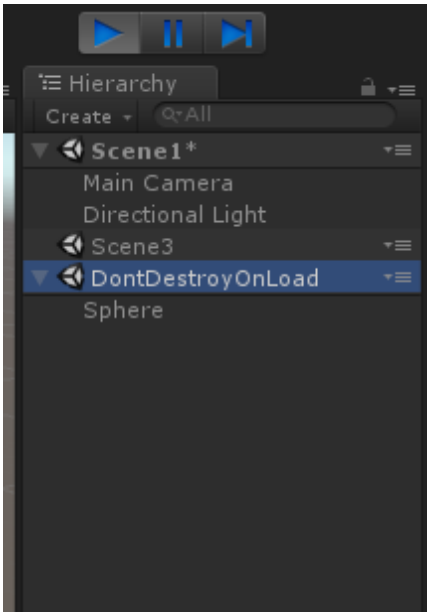
当前场景的物体逐个被销毁 Destroy

新场景的物体逐个被实例化 Instantiate

场景中会有一些必须一直被保存 即使场景切换也不能被销毁的物体

```
void Awake () {
      //当场景切换时 不销毁gameObject
```

```
        DontDestroyOnLoad(gameObject);
}
```



场景的淡入淡出

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

public class Load : MonoBehaviour {

    //黑色图片
    public Texture texture;
    //默认UI颜色 只修改其A值
    private Color color=Color.white;
    //淡入或淡出的消耗时间
    private float fadeTime = 0.5f;
    //用来更新的当前时间
    private float time;
    //是否需要淡出
    private bool fadeOut;
    //是否需要淡入
    private bool fadeIn;

    //private int index;
```

```
    //private string sceneName;
    void Start () {
        //不能切换场景时销毁 因为到新场景需要淡入功能
        DontDestroyOnLoad(gameObject);
        color.a = 0f;//初始透明度为0  完全透明
    }

    void Update()
    {
        if (fadeOut)
        {
            FadeOut();
        }

        if (fadeIn)
        {
            FadeIn();
        }
    }

    //淡出场景的方法
    void FadeOut()
    {
        time += Time.deltaTime;
        //Alpha值为0时 完全透明 Alpha为1时 完全不透明(原本图片的透明度)
        color.a = time / fadeTime;//time从0到2的变化区间
        if (time > fadeTime)
        {
            fadeOut = false;
        }
    }

    //淡入场景的方法
    void FadeIn()
    {
        time -= Time.deltaTime;
        color.a = time / fadeTime;//time从2到0的变化区间
        if (time < 0)
        {
            fadeIn = false;
        }
    }
```

```
64
65      void OnGUI () {
66          GUI.color = Color.white;
67          if (GUILayout.Button("切换到场景2"))
68          {
69              //当点击按钮切换场景时 首先要做到淡出
70              fadeOut = true;
71              //场景切换要延迟到淡出效果结束 fadeTime
72              Invoke("LoadSceneDelay", fadeTime);
73          }
74          //把实时更新的透明度赋值给UI
75          GUI.color = color;
76
77          GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
    texture);
78
79      }
80
81      //public void LoadSceneByFade(int buildIndex)
82      //{
83      //    index = buildIndex;
84      //    Invoke("LoadSceneDelay", fadeTime);
85      //}
86      //public void LoadSceneByFade(string name)
87      //{
88      //    sceneName= name;
89      //    Invoke("LoadSceneDelay", fadeTime);
90      //}
91
92      void LoadSceneDelay()
93      {
94          //延迟时间到 新场景切换
95          SceneManager.LoadScene("Scene2");
96          //切换到新场景 开始淡入
97          fadeIn = true;
98          time = fadeTime;
99      }
100 }
101
```

Unity实现定时器(延时功能)

1 Time.deltaTime

```
    private float time;
    private float delayTime=3.0f;

    void Update()
    {
        time += Time.deltaTime;
        if (time > delayTime)
        {
            Debug.Log("3秒时间到");
            time = 0;
        }
    }
```

2 使用Invoke和InvokeRepeating实现延时功能

```
private float delayTime=1.0f;
    private int count;

    void Start()
    {
        //Invoke("Fn", delayTime);
        InvokeRepeating("Fn1", delayTime, delayTime);
    }

    public void Fn1()
    {
        Debug.Log("1.0秒时间到");
        count++;
        if (count >= 3)
        {
            //判断Fn1方法 是不是在Invoke语法的循环中
            if (IsInvoking("Fn1"))
            {
                //取消延迟功能
                CancelInvoke("Fn1");
            }
        }
```

```
    }
```

3 协同程序 协程

```
public class UI : MonoBehaviour {
    void Start()
    {
        //调用方法的形式开启协程
        StartCoroutine(Fun());
        //即使调用了Stop 也无法停止
        //StopCoroutine(Fun());
        //字符串形式开启协程
        StartCoroutine("Fun");
        StopCoroutine("Fun");

        //字符串形式和方法形式 都可以被停止
        StopAllCoroutines();
    }

    IEnumerator Fun()
    {
        int x = 100;
        //等待1秒时间到
        yield return new WaitForSeconds(1.0f);
        x++;
        Debug.Log(x);

        //等待2秒时间到
        yield return new WaitForSeconds(2.0f);
        x++;
        Debug.Log(x);

        //延迟一帧  返回int值是延迟一帧 和int具体的数据无关
        yield return 0;
        yield return null;
    }
}
```

多线程开发再Unity等很多游戏引擎中使用 有很大的限制

（在分线程中无法使用UnityEngine的API）

Unity提供了基于时间分割策略的协同程序

协程可以理解为多线程的替代方案

和yield return 语法的联用可以实现很多类似多线程的功能

协程的执行流程：

协程运行在每一帧的Update之后

协程的代码也是逐条执行 遇到yield return语法后 会去判断后续的条件

（延迟1帧 延迟一段时间 等待加载完成 等待任务完成等）

当条件不满足时 协程会被挂起 并记录协程内部的执行状态(执行位置 内部局部变量等)

在下一次进入协程时 会继续上一次的执行状态往下执行和判断

当条件满足时 协程直接往下执行

当后续没有更多的yield return 语法时 协程会在一帧执行完