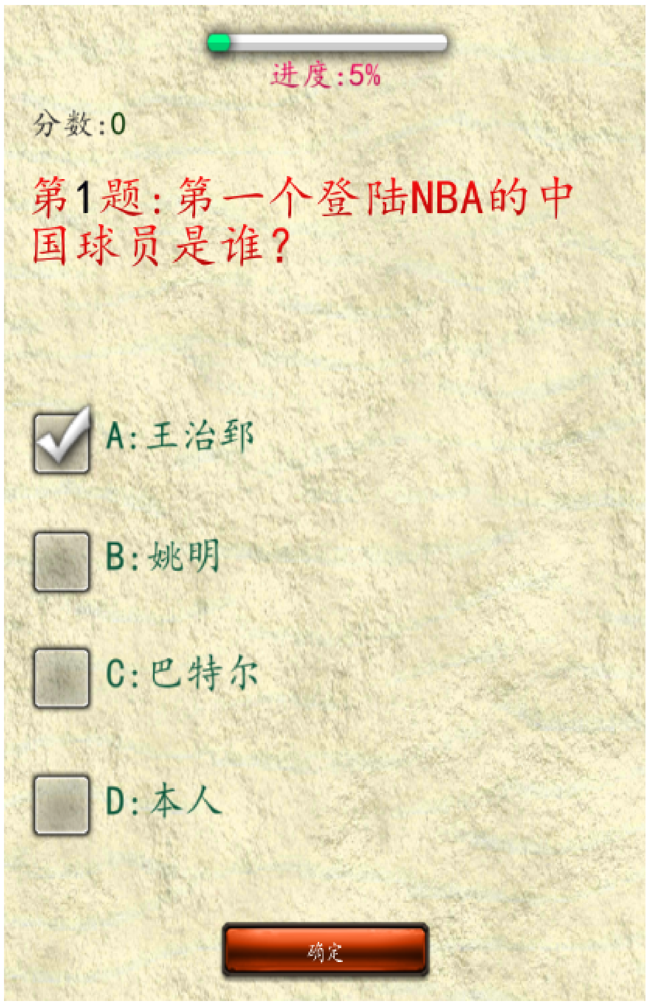
基于NGUI的答题系统



XML配置文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<QuestionConfig>
    <Question Desc="第一个登陆NBA的中国球员是谁？ " RightKey="1">
        <Option>A:王治郅</Option>
        <Option>B:姚明</Option>
        <Option>C:巴特尔</Option>
        <Option>D:本人</Option>
    </Question>
    <Question Desc="2002年奥运会举行城市是在哪里？ " RightKey="4">
        <Option>A:雅典</Option>
        <Option>B:北京</Option>
        <Option>C:洛杉矶</Option>
        <Option>D:扯呢？ 2002年哪有奥运会</Option>
    </Question>
    <Question Desc="2006年男足世界杯最终夺冠的是哪个国家？ " RightKey="4">
        <Option>A:德国</Option>
        <Option>B:巴西</Option>
        <Option>C:中国</Option>
        <Option>D:意大利</Option>
    </Question>
    <Question Desc="《权利的游戏》中的角色提利昂·兰尼斯特的昵称是？" 
RightKey="3">
        <Option>A:小扒皮</Option>
        <Option>B:弑君者</Option>
        <Option>C:小恶魔</Option>
        <Option>D:北境之王</Option>
    </Question>
    <Question Desc="被誉为中国最美五大湖泊之首的是？" RightKey="1">
        <Option>A:青海湖</Option>
        <Option>B:喀纳斯湖</Option>
        <Option>C:纳木措</Option>
        <Option>D:西湖</Option>
    </Question>
    <Question Desc="中国西部经济最发达的城市是？" RightKey="2">
        <Option>A:拉萨</Option>
        <Option>B:西宁</Option>
        <Option>C:乌鲁木齐</Option>
        <Option>D:林芝</Option>
    </Question>
    <Question Desc="下列所列举的河流哪一组全部是自东向西流的河流？" 
RightKey="2">
        <Option>A:黄河,倒淌河,额尔齐斯河</Option>
        <Option>B:倒淌河,伊犁河,塔里木河</Option>
        <Option>C:淮河,伊犁河,雅鲁藏布江下游</Option>
        <Option>D:伊犁河,雅鲁藏布江下游,黄河</Option>
    </Question>
    <Question Desc="黄河的发源地是在哪一个山脉？" RightKey="1">
        <Option>A:巴颜喀拉山脉</Option>
        <Option>B:昆仑山脉</Option>
        <Option>C:祁连山脉</Option>
        <Option>D:喜马拉雅山山脉</Option>
    </Question>
    <Question Desc="C#中修饰常量并且可以在构造函数中赋值的关键字是？" 
RightKey="2">
        <Option>A:const</Option>
        <Option>B:readonly</Option>
        <Option>C:vritual</Option>
        <Option>D:ref</Option>
    </Question>
    <Question Desc="C#中的interface是为了解决什么问题而存在的？" RightKey="2">
        <Option>A:虚继承</Option>
        <Option>B:多重继承</Option>
        <Option>C:父类和子类的通信</Option>
        <Option>D:子类向父类构造传参</Option>
    </Question>
    <Question Desc="下面哪个不是面向对象的特征之一？" RightKey="3">
        <Option>A:多态</Option>
        <Option>B:继承</Option>
        <Option>C:友元</Option>
        <Option>D:封装</Option>
    </Question>
    <Question Desc="C#中实现函数重载的情况不包括？" RightKey="3">
        <Option>A:参数的类型不同</Option>
        <Option>B:参数的个数不同</Option>
        <Option>C:参数的名字不同</Option>
        <Option>D:参数的先后顺序不同</Option>
    </Question>
    <Question Desc="unity中的灯光类型不包括？" RightKey="4">
        <Option>A:Directional</Option>
        <Option>B:Point</Option>
        <Option>C:Spot</Option>
        <Option>D:Diffuse</Option>
    </Question>
    <Question Desc="unity周期函数的调用顺序是？" RightKey="1">
        <Option>A:Awake->OnEnable->Start</Option>
        <Option>B:OnEnable->Awake->Start</Option>
        <Option>C:Start->Awake->OnEnable</Option>
        <Option>D:Start->OnEnable->Awake</Option>
    </Question>
    <Question Desc="对yield return的正确理解是？" RightKey="2">
        <Option>A:等待一帧后继续执行</Option>
        <Option>B:等待条件满足后继续执行</Option>
        <Option>C:等待其它线程执行完以后执行</Option>
        <Option>D:等待条件满足后重新执行</Option>
    </Question>
    <Question Desc="以下描述正确的是" RightKey="4">
        <Option>A:string类型是值类型</Option>
        <Option>B:object类型是值类型</Option>
        <Option>C:枚举是引用类型</Option>
        <Option>D:结构体是值类型</Option>
    </Question>
    <Question Desc="对virtual关键字的描述正确的是？" RightKey="3">
        <Option>A:virtual修饰抽象类</Option>
        <Option>B:virtual是对虚函数的修饰</Option>
        <Option>C:virtual修饰派生类</Option>
        <Option>D:virtual只能内部访问</Option>
    </Question>
    <Question Desc="C#中每个int 类型的变量占用__个字节的内存,占用__位二进制" RightKey="1">
        <Option>A:4,32</Option>
        <Option>B:4,8</Option>
        <Option>C:2,32</Option>
        <Option>D:2,8</Option>
    </Question>
    <Question Desc="UI开发中对界面的适配理解错误的是？" RightKey="4">
        <Option>A:基于设计分辨率对UI进行缩放</Option>
        <Option>B:UIRoot的缩放模式一般基于高度的</Option>
        <Option>C:当UI显示不完整可以调整Camera的Size</Option>
        <Option>D:不用考虑显示内容是否超出屏幕</Option>
    </Question>
    <Question Desc="下面碰撞器中不一定是2D碰撞器的是" RightKey="2">
        <Option>A:EdgeCollider</Option>
        <Option>B:BoxCollider</Option>
        <Option>C:CircleCollider</Option>
        <Option>D:PolygonCollider</Option>
    </Question>
</QuestionConfig>
```

XML读取

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

public class Question
{
    public int rightKey;
    public string desc;
    public List<string> options = new List<string>();

    public Question(XmlElement element)
    {
        desc = element.GetAttribute("Desc");
        rightKey = int.Parse(element.GetAttribute("RightKey"));
        foreach (XmlElement data in element.ChildNodes)
            options.Add(data.InnerText);
    }
}

public class QuestionConfig{

    private static QuestionConfig instance;
    public static QuestionConfig Instance
    {
        get{
            if (instance == null)
                instance = new QuestionConfig();
            return instance;
        }
    }

    public List<Question> questions = new List<Question>();

    public void LoadXml()
    {
        XmlDocument xml = new XmlDocument();
        xml.Load(Application.dataPath + "/QuestionConfig.xml");

        XmlElement node = xml.DocumentElement;

        foreach (XmlElement element in node)
        {
            Question question = new Question(element);
            questions.Add(question);
        }
    }
}
```

UI界面

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

public class UIGameMain : MonoBehaviour {

    public UIProgressBar progress;
    public UILabel sliderLabel;
    public UILabel questionLabel;
    public UILabel scoreText;
    public List<UIToggle> options;
    public List<UILabel> optionTexts;

    private float questionCount;
    private float currentCount;
    private int score;
    private Question currentQuestion;

    void Start () {
        QuestionConfig.Instance.LoadXml();
        questionCount = QuestionConfig.Instance.questions.Count;
        currentCount = 1;
        score = 0;
        ChangeQuestion();
        ChangeUIData();
    }

    void ChangeQuestion()
    {
        currentQuestion =
QuestionConfig.Instance.questions[(int)currentCount - 1];
        questionLabel.text = string.Format("第[0000ff]{0}[-]题:{1}",
currentCount, currentQuestion.desc);

        for (int i = 0; i < optionTexts.Count; i++)
        {
            optionTexts[i].text = currentQuestion.options[i];
        }
    }

    void ChangeUIData()
    {
        scoreText.text = string.Format("分数:[00ff00]{0}[-]", score);

        float x = currentCount / questionCount;

        if (currentCount <= questionCount)
        {
            sliderLabel.text = string.Format("进度:{0}%", (int)(x * 
100));
            progress.value = x;
        }
    }


    public void OnClick()
    {
        if (currentCount > questionCount)
            return;

        for (int i = 0; i < options.Count; i++)
        {
            if (options[i].value)
            {
                if (currentQuestion.rightKey == i + 1)
                {
                    Debug.Log("答对了");
                    score += 100;
                }
                else
                {
                    Debug.Log("答错了");
                }
                break;
            }
        }

        currentCount++;
        if (currentCount > questionCount)
        {
            Debug.Log("您已经完成所有的题目 总分是" + score.ToString());
            ChangeUIData();
            return;
        }

        ChangeQuestion();
        ChangeUIData();
    }
}
```

插值 Lerp

在两个点之间 插值计算（比例运算）

(a,b,t)

a,b表示两个坐标 t表示间隔 取值范围是0-1

若t为0 返回 a 若t为1 返回 b 若t为0.5 返回a和b的中间的点

```
    void Start () {
        Vector3 v= Vector3.Lerp(Vector3.zero, new Vector3(100, 100, 100), 0f);
        Debug.Log(v);
        Vector3 v2 = Vector3.Lerp(Vector3.zero, new Vector3(100, 100, 100), 0.5f);
        Debug.Log(v2);
        Vector3 v3 = Vector3.Lerp(Vector3.zero, new Vector3(100, 100, 100), 2f);
        Debug.Log(v3);
    }
```

使用插值控制人物移动

3秒钟移动到目标点

```
public class LerpText : MonoBehaviour {

    public Transform cube1;

    private Vector3 start;


    private float time;

    void Start()
    {
        start = transform.position;
    }
    void Update () {
        time += Time.deltaTime;
        transform.position = Vector3.Lerp(start, cube1.position, time / 3.0f);
    }
}
```

标准摄像机跟随代码 无弹簧效果

```
public class LerpText : MonoBehaviour {

    public Transform player;

    private Vector3 standardDirection;

    void Start()
    {
        standardDirection = player.position - transform.position;
    }
    void Update () {
        transform.position = player.position - standardDirection;
    }
}
```

利用插值实现弹簧跟随效果

```
public class LerpText : MonoBehaviour {

    public Transform player;

    private Vector3 standardDirection;

    void Start()
    {
        standardDirection = player.position - transform.position;
    }
    void Update () {
        Vector3 pos = player.position - standardDirection;

        transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime*3.0f);
    }
}
```

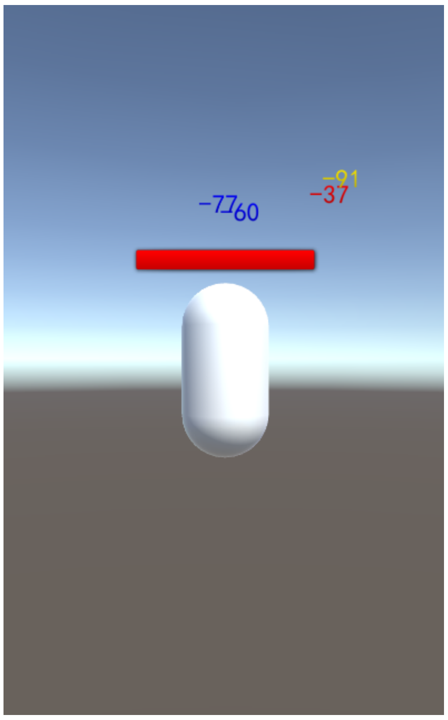
球形插值

```
public class LerpText : MonoBehaviour {

    public Transform cube1;
    public Transform cube2;
    public Transform cube3;
    void Update () {
        for (int i = 0; i <=10; i++)
        {
            Debug.DrawLine(cube3.position, Vector3.Slerp(cube1.position, cube2.position, i * 0.1f), Color.red);
        }
    }
}
```

颜色插值

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

public class LerpText : MonoBehaviour {

    private UISprite sprite;

    private float time;

    private Color startColor;
    void Start()
    {
        sprite = GetComponent<UISprite>();
        startColor = Color.red;
        startColor.a = 0;
    }
    void Update () {
        time += Time.deltaTime;
        sprite.color = Color.Lerp(startColor, Color.red, time / 3.0f);
        //transform.localScale = Vector2.Lerp(Vector2.zero, 
Vector2.one, time / 3.0f);
    }
}

```

人物血条功能实现



绑定在3D物体身上

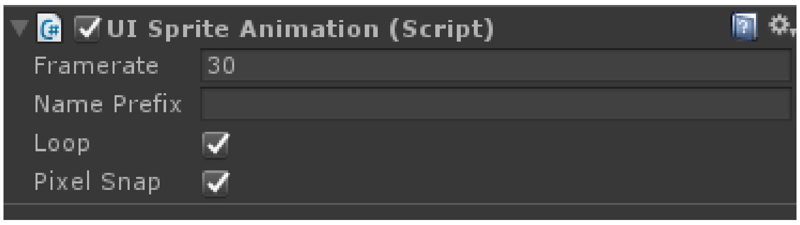
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Text : MonoBehaviour {

    public Transform bloodUI;
    public Transform blood;

    private float defaultSize;


    public GameObject hurtItem;
    void Start () {
        defaultSize = Vector3.Distance(Camera.main.transform.position, 
blood.position);
    }

    void Update () {
        //控制UI缩放 如果血条大小永远不变 则不需要修改UI的Scale
        //原理：基于UI点到摄像机的距离 求得比例值

        float newSize = 
Vector3.Distance(Camera.main.transform.position, blood.position);

        if(UICamera.currentCamera!=null)
            bloodUI.position = Convert(blood.position);

        bloodUI.localScale = Vector3.one * (defaultSize/newSize);

        if (Input.GetMouseButtonDown(0))
        {
            Hurt();
        }

    }

    Vector3 Convert(Vector3 point)
    {
        //原理：多个摄像机同时存在时 因为摄像机位置的不同 世界坐标不一致 但是屏幕坐标是相同的
        //先通过主摄像机把世界坐标转化为屏幕坐标
        Vector3 pos= Camera.main.WorldToScreenPoint(point);
        //通过UI摄像机 把屏幕坐标转换为世界坐标
        Vector3 ui_pos=UICamera.currentCamera.ScreenToWorldPoint(pos);
        ui_pos.z = 0f;
        return ui_pos;
    }

    void Hurt()
    {
        GameObject obj=Instantiate(hurtItem);
        Destroy(obj, 2.0f);
        int hurt=Random.Range(30,100);
        obj.GetComponent<Hurt>().ChangeData(hurt, bloodUI.position);
        obj.transform.parent = bloodUI;
        obj.transform.localScale = Vector3.one;

    }
}

```

绑定在掉血的预设上

```

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

public class Hurt : MonoBehaviour {

    private Color[] colors = new Color[] { Color.white, Color.red, Color.yellow, Color.green, Color.blue };

    public UILabel label;

    private Vector3 targetPosition;

    private float time;

    private Vector3 startPosition;

    public void ChangeData(int hurt,Vector3 postion) {

        label.text = "-" + hurt.ToString();
        label.color = colors[Random.Range(0, colors.Length)];
        transform.position = postion;
        startPosition = postion;
        targetPosition=postion+new
Vector3(Random.Range(-0.1f,0.1f),Random.Range(-0.1f,0.1f),0f)*5f;
    }


    void Update () {
        time += Time.deltaTime;
        transform.position = Vector3.Lerp(startPosition,
targetPosition, time / 2.0f);

    }
}

```

Tween

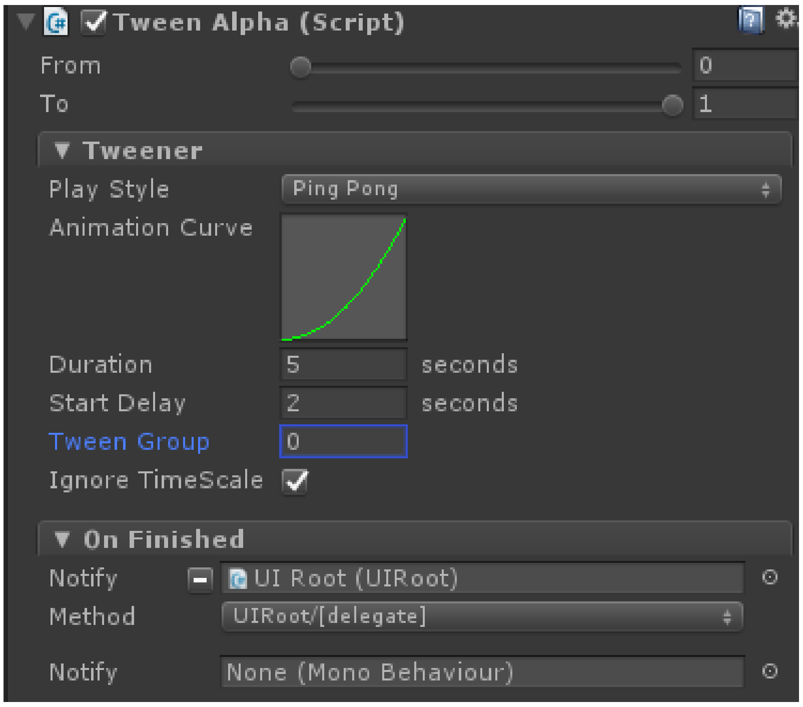
精灵动画 SpriteAnimation


UI Sprite Animation (Script)

Framerate	30
Name Prefix	
Loop	✓
Pixel Snap	✓

NGUI提供的一些动画效果

TweenAlpha 透明度渐变



Tween Alpha (Script)

From 0

To 1

Tweener

Play Style: Ping Pong

Animation Curve

Duration: 5 seconds

Start Delay: 2 seconds

Tween Group: 0

Ignore TimeScale: ✓

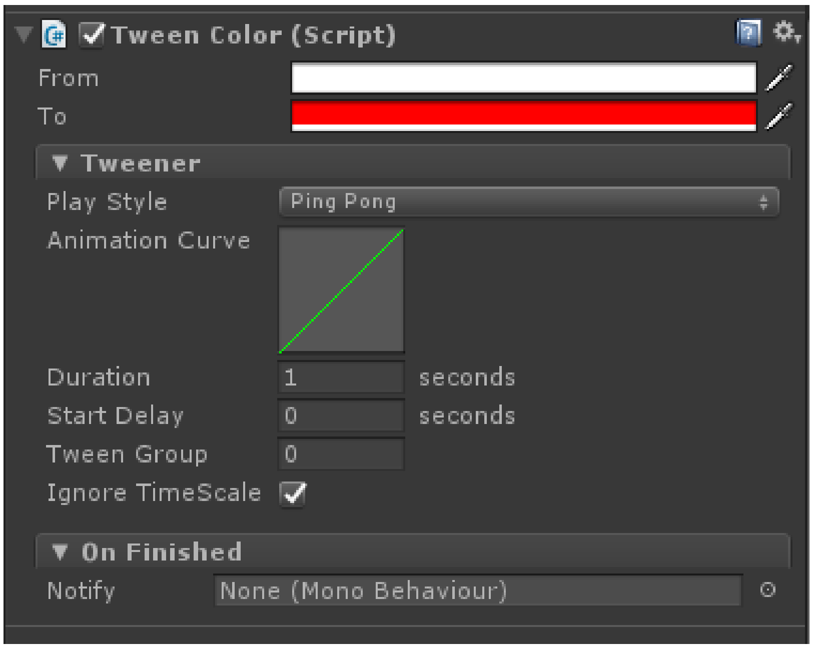
On Finished

Notify: UI Root (UIRoot)

Method: UIRoot/[delegate]

Notify: None (Mono Behaviour)

TweenColor 颜色渐变



Tween Color (Script)

From

To

Tweener

Play Style: Ping Pong

Animation Curve

Duration: 1 seconds

Start Delay: 0 seconds

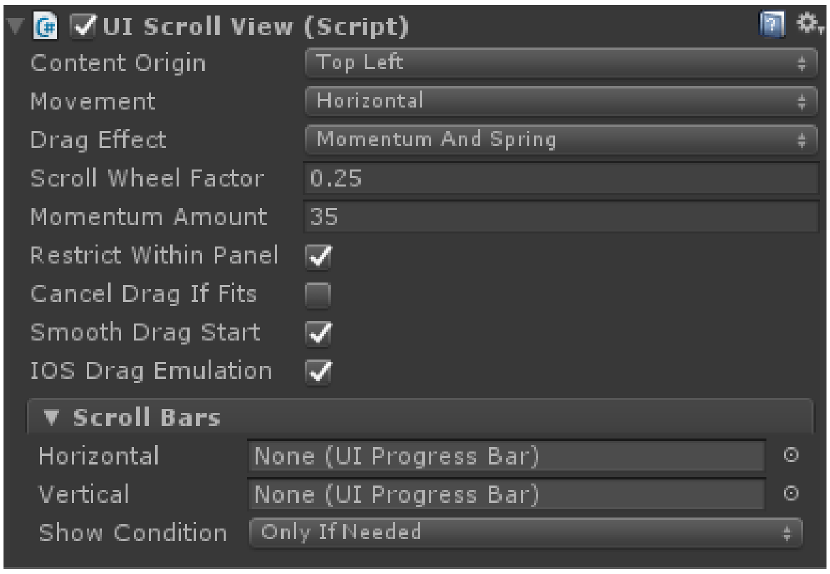
Tween Group: 0

Ignore TimeScale: ✓

On Finished

Notify: None (Mono Behaviour)

ScrollView


UI Scroll View (Script)

Content Origin	Top Left
Movement	Horizontal
Drag Effect	Momentum And Spring
Scroll Wheel Factor	0.25
Momentum Amount	35
Restrict Within Panel	✓
Cancel Drag If Fits	
Smooth Drag Start	✓
IOS Drag Emulation	✓

Scroll Bars

Horizontal	None (UI Progress Bar)
Vertical	None (UI Progress Bar)
Show Condition	Only If Needed

ContentOrigin

控制Panel相对ScrollView的位置

Movement 运动模式

水平 垂直 随意滑动 自定义

DragEffect

拖动效果 Momentun And Spring 拖到边界松开时拖拽会有弹回效果

ScrollWheelFactor

鼠标滚轮速度 0：滚轮不生效

Momentum Amount

滑动后自动滑动距离

Restrict Within Panel

Scrollview不会滑出Panel

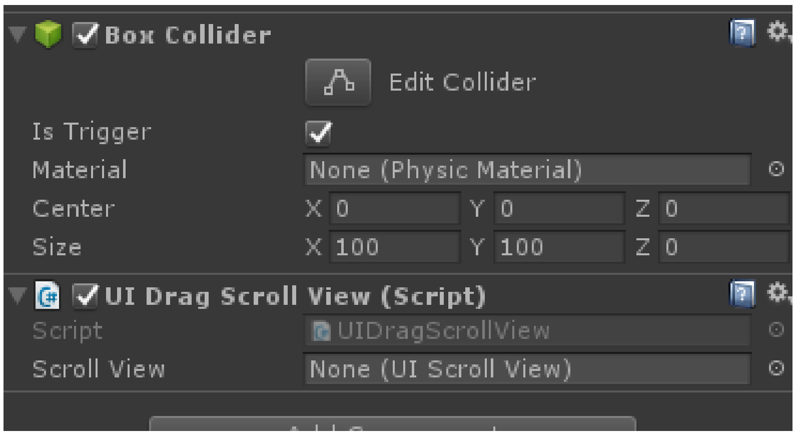
Cancel Drag if Fits

当适合视窗时 自动退出拖动

Smooth Drag Start   IOS Drag Emulation

模拟IOS手机系统的拖动效果

ScorllView下所有的Item 必须携带触发器和UIDragScrollView组件



Box Collider

Edit Collider

Is Trigger ✓

Material None (Physic Material)

Center X 0 Y 0 Z 0

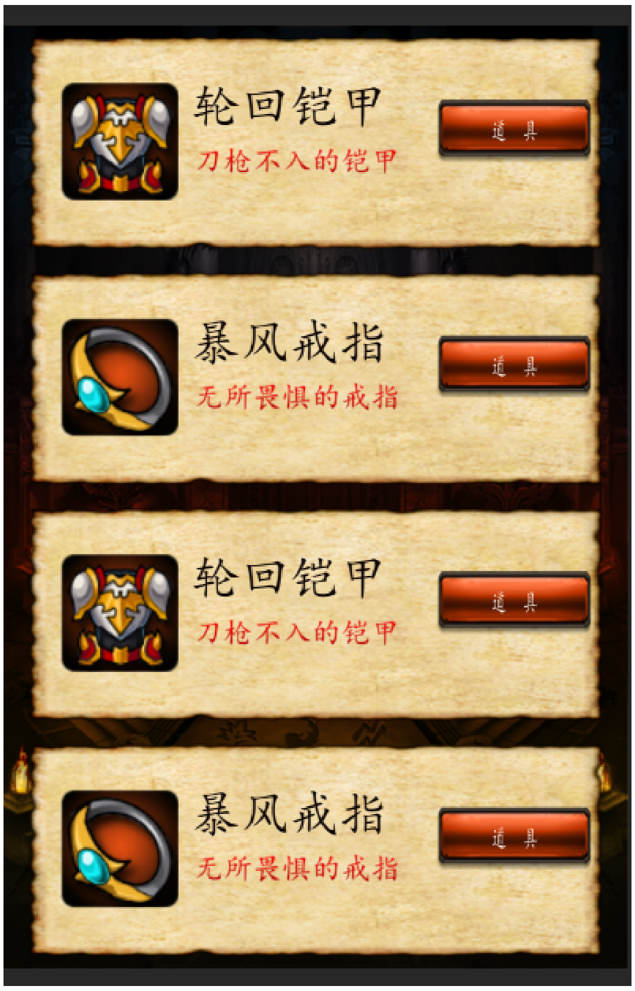
Size X 100 Y 100 Z 0

UI Drag Scroll View (Script)

Script UIDragScrollView

Scroll View None (UI Scroll View)

实现背包功能



在动态添加元素到Grid下面后 Grid重新排序功能

UIGrid.Reposition()