

# Unity 3D NGUI 实战教程

高雪峰 编著

本书获得以下专业社区一致推荐



**UI中国** UI.cn

中国排名第一的专业界面设计社区



**GAMEUI** Gameui.com

中国专精于游戏视觉相关的设计分享门户网站



**icon** icon.cn

中国专业的游戏 UI 设计咨询机构



人民邮电出版社  
POSTS & TELECOM PRESS

封面设计 / icon.cn



**90%** 的智能手机 3D 游戏使用了 Unity 引擎

而 **90%** 的 Unity 游戏使用了 NGUI 制作 UI

这是 **国内第一本** 讲解 NGUI 设计的实战教程

分类建议: 计算机 / 移动开发  
计算机 / UI 设计

人民邮电出版社网址: [www.ptpress.com.cn](http://www.ptpress.com.cn)



ISBN 978-7-115-38546-8



ISBN 978-7-115-38546-8

定价: 49.00 元



# Unity 3D NGUI 实战教程

高雪峰 编著

人民邮电出版社  
北 京

## 图书在版编目 (C I P) 数据

Unity 3D NGUI实战教程 / 高雪峰编著. -- 北京 :  
人民邮电出版社, 2015.3  
ISBN 978-7-115-38546-8

I. ①U… II. ①高… III. ①游戏程序—程序设计—  
教材 IV. ①TP311.5

中国版本图书馆CIP数据核字(2015)第036349号

## 内 容 提 要

NGUI 是专门针对 Unity 引擎、用 C#语言编写的一套插件,它已经成为了目前世界上应用最广、最成熟的 Unity 制作 UI 的插件,完美地弥补了 Unity 引擎原生 GUI 系统和 NewGUI 系统的各种不足。程序员可以利用它提供的一整套 UI 框架和事件通知系统来进行自己项目的 UI 设计和制作。本书不仅讲解了必知必会的 NGUI 的基础知识,更是以项目实战为目的,涵盖了大量的项目实战中的经验之谈和技巧总结,用以帮助读者达到学以致用目的。

本书的主要内容:初识 NGUI、UI 开发的流程、NGUI 强大优势、制作第一个 UI 图集、创建一个 3D UI、查看和管理 UI 的深度、制作基础的 UI 控件、让 UI 动起来——UI 动画、NGUI 进阶、使用 Panel 管理面板、NGUI 实战进阶、UI 开发核心问题——UI 随屏幕自适应、实战开发中 UI 资源制作标准、跨平台制作 UI 资源、UI 结构设计、UI 代码的设计和优化、项目案例实战分析、背包界面的制作等核心技术,最后用一章归纳了 NGUI 常见疑难问题,以便读者遇到问题时可以随时参考。

本书适合新上手的 Unity 客户端程序员、需要做 UI 的 Unity 程序员、想自学 Unity 做独立游戏开发的人员,以及大专院校相关专业的师生学习用书和培训学校的教材。

- 
- ◆ 编 著 高雪峰  
责任编辑 张 涛  
责任印制 张佳莹 焦志炜
  - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号  
邮编 100164 电子邮件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>  
北京天宇星印刷厂印刷
  - ◆ 开本: 800×1000 1/16  
印张: 15  
字数: 358 千字 2015 年 3 月第 1 版  
印数: 1—3 000 册 2015 年 3 月北京第 1 次印刷
- 

定价: 49.00 元

读者服务热线: (010)81055410 印装质量热线: (010)81055316  
反盗版热线: (010)81055315



# 前言

在手机游戏开发兴起的当下，Unity 3D 引擎依靠其良好的跨平台特性，一跃成为全球第一大引擎，被广泛地使用。越来越多的游戏开发者开始关注和使用 Unity 3D 引擎。Unity 3D 引擎最大的短板在于其原生的 GUI 系统有很大的缺陷，例如，性能和方便程度等都不适合进行商业开发，所以，大部分开发者都开始使用 NGUI。NGUI 的 GUI 以良好的性能优化、方便的开发模式、成熟稳定等特点，已经成为全球 Unity 游戏开发者的 UI 制作首选插件。因为 NGUI 是一个插件的缘故，网上很难有成熟的资料，即便要查找一些基本的资料也得连入国外的网站，而且是英文的资料，这给开发者带来了很大的苦恼。由于 UI 开发是游戏开发中客户端的重头工作，具有责任大、工作量多、修改频繁等诸多特性，一直是客户端程序员的一个疑难问题。现在市面上还没有 NGUI 的书，为了让读者能够学会 NGUI 的用法和技巧，并且能够直接运用于正式的商业项目开发中，作者结合自己的实践经验特意撰写了本书。

## 本书的特点

本书不仅讲解了必知必会的基础知识，更是以项目实战为目的，书中涵盖了大量的项目实战中的经验之谈和技巧总结，这对于一个客户端程序员来说，不管他用什么引擎、用什么 UI 工具来开发 UI 系统，本书都能给他提供帮助，达到学以致用用的目的。

## 本书的主要内容

本书全面讲解了 NGUI 的实战知识，主要内容为：初识 NGUI、UI 开发的流程、NGUI 强大优势、导入 NGUI 插件、认识 UI 的基本资源、制作第一个 UI 图集、用 AtlasMaker 制作图集、制作第一个 UI 字体、创建一个 3D UI、3D UI 的工作原理、查看和管理 UI 的深度、制作基础的 UI 控件、精灵的创建、制作 UI 纹理、制作按钮、制作进度条、制作滑动条、制作输入框、制作滚动视图、制作复选框、让 UI 动起来——UI 动画、颜色变化动画、位置变化动画、旋转变换动画、大小变化动画、组件整体变换、音量变化动画、在 UI 中使用 Animation 动画、动画控制——UIPlayTween 组件、NGUI 进阶、使用 Panel 管理面板、使用 Grid 排列元素、使用 Toggle 制作页签、使用 DragCamera 直接拖动摄像机、使用 DragObject 直接拖动物体、拖动

改变 UI 元素的尺寸、按钮绑定快捷键、制作列表、打字机慢慢出字的效果、NGUI 实战进阶、UI 开发核心问题——UI 随屏幕自适应、背景图的适配、UI 元素的相对自适应、多个摄像机同时协作运行、实战开发中 UI 资源制作标准、跨平台制作 UI 资源、巧用九宫格减少 UI 资源量、UI 事件监听的遮挡、NGUI 和模型、特效在同一层中混用、UI 结构设计、用代码操作 NGUI 的类、获取 NGUI 的组件、迅速判断类中可读写的成员、动态创建 UI 元素、Sprite 的常用操作、动态对 Button 设置单击事件、网格动态增减成员和刷新排列、手动控制动画随意播放、调用进度条、巧用 EventTrigger 监听各种事件、UI 代码的设计和优化、项目案例实战分析、场景加载的进度条界面制作、RPG 游戏中人物头像状态栏的制作、技能快捷栏的制作、RPG 角色头顶跟随血条的制作、背包界面的制作等核心技术，最后用一章归纳了 NGUI 常见疑难问题，以便读者遇到问题时可以随时参考。

## 本书作者

高雪峰，现为游戏制作人，曾经担任游戏主策划、游戏运营、Unity 程序员等职位，开发过端游、页游、手游等项目，带团队做过多个商业项目，对游戏的研发过程具有丰富的经验和实战技能。对 NGUI 有深入的研究，并且全部应用于项目实战，本书是作者多年实战经验的总结，定会给读者带来很多有益的实战启示。

## 本书读者对象

本书适合新上手的 Unity 客户端程序员、需要做 UI 的 Unity 程序员、想自学 Unity 做独立游戏开发的人员阅读，也适合用作大专院校相关专业的师生学习用书和培训学校的教材。

编辑联系邮箱：[zhangtao@ptpress.com.cn](mailto:zhangtao@ptpress.com.cn)。



# 目 录



## 第 1 章 初识 NGUI.....1

### 1.1 游戏 UI 开发介绍.....1

#### 1.1.1 什么是游戏 UI.....1

#### 1.1.2 UI 为何如此重要.....1

#### 1.1.3 UI 开发的流程.....2

#### 1.1.4 UI 开发的难点.....2

### 1.2 什么是 NGUI.....3

#### 1.2.1 NGUI 插件介绍.....3

#### 1.2.2 NGUI 的强大优势.....3



## 第 2 章 NGUI 基础.....5

### 2.1 导入 NGUI 插件.....5

#### 2.1.1 NGUI 版本介绍.....5

#### 2.1.2 NGUI 的下载和购买.....5

#### 2.1.3 导入 NGUI 插件应用.....6

#### 2.1.4 导入常见问题.....9

### 2.2 认识基本的 UI 资源.....10

#### 2.2.1 什么是 UI 精灵 (Sprite).....10

#### 2.2.2 什么是 UI 图集 (Atlas).....10

#### 2.2.3 什么是 UI 贴图 (Texture).....10

#### 2.2.4 什么是 UI 标签 (Label).....12

#### 2.2.5 什么是 UI 字体 (Font).....12

### 2.3 制作第一个 UI 图集.....13

#### 2.3.1 学会解剖 UI 的资源结构.....13

#### 2.3.2 如何导入切好的美术资源.....15

#### 2.3.3 用 Atlas Maker 制作图集.....16

### 2.4 制作第一个 UI 字体.....20

#### 2.4.1 为什么要制作 UI 字体.....20

#### 2.4.2 静态字体和动态字体.....20

#### 2.4.3 制作静态字体介绍.....21

#### 2.4.4 制作动态字体介绍.....22

### 2.5 创建第一个 UI.....22

#### 2.5.1 创建一个 2D UI.....22

#### 2.5.2 创建一个 3D UI.....24

#### 2.5.3 了解 UIRoot、UIPanel 和 UICamera 组件.....24

### 2.6 2DUI 和 3DUI 的工作原理.....28

#### 2.6.1 2DUI 的工作原理.....28

#### 2.6.2 3DUI 的工作原理.....30

#### 2.6.3 如何判断该选择哪一种 UI.....31

### 2.7 深度 (Depth) 概念.....31

#### 2.7.1 强化对深度的理解.....31

#### 2.7.2 小心相机的深度.....32



### 第3章 核心组件.....34

- 3.1 什么是 UI 控件 .....34
- 3.2 制作精灵 (UISprite) .....34
  - 3.2.1 怎样判断是否应该使用精灵.....34
  - 3.2.2 创建精灵.....35
  - 3.2.3 Sprite 组件的设置 .....37
- 3.3 制作标签 (Label) .....43
  - 3.3.1 怎样判断是否应当使用标签.....43
  - 3.3.2 创建标签.....43
  - 3.3.3 Label 的文字设置.....43
- 3.4 制作 UI 纹理 (UITexture) .....46
  - 3.4.1 什么情况下使用 UITexture.....46
  - 3.4.2 创建纹理.....46
  - 3.4.3 纹理的设置 .....46
- 3.5 制作按钮 (Button) .....48
  - 3.5.1 怎样判断应该使用按钮.....48
  - 3.5.2 创建按钮.....49
  - 3.5.3 核心组件 BoxCollider .....49
  - 3.5.4 核心组件 UIButton .....52
  - 3.5.5 制作按钮的放缩动画 .....54
  - 3.5.6 制作按钮的偏移动画 .....55
  - 3.5.7 制作按钮的旋转动画 .....56
  - 3.5.8 添加按钮单击音效 .....56
  - 3.5.9 任何事物都可以变成按钮,  
不仅仅是 UI.....57
- 3.6 制作进度条 (UISlider) .....58
  - 3.6.1 怎样判断是否应当使用  
进度条.....58
  - 3.6.2 创建进度条.....59

- 3.6.3 核心组件 UISlider 设置 .....60
- 3.6.4 进度条的 BoxCollider 说明.....62
- 3.7 制作输入框 (Input) .....63
  - 3.7.1 怎样判断是否应当使用  
输入框.....63
  - 3.7.2 创建输入框.....63
  - 3.7.3 核心组件 Input 设置.....64
  - 3.7.4 输入框使用的一些  
注意事项 .....67
- 3.8 制作滚动视图 (ScrollView) .....68
  - 3.8.1 怎样判断是否应当使用  
滚动视图 .....68
  - 3.8.2 创建滚动视图 .....69
  - 3.8.3 滚动视图核心组件  
UIPanel .....69
  - 3.8.4 滚动视图核心组件  
UIScrollView .....72
  - 3.8.5 创建一个拖动条 .....75
  - 3.8.6 拖动条说明 .....76
  - 3.8.7 让视图内的内容可以  
被拖动 .....77
  - 3.8.8 制作滚动视图时的  
注意事项 .....78
- 3.9 制作复选框 (Toggle) .....79
  - 3.9.1 怎样判断是否应当使用  
复选框 .....79
  - 3.9.2 创建复选框 .....79
  - 3.9.3 复选框的核心组件  
UIToggle .....80
- 3.10 制作下拉菜单 (PopupMenu) .....82
  - 3.10.1 怎样判断是否应当使用  
下拉菜单 .....82
  - 3.10.2 创建下拉菜单 .....82
  - 3.10.3 显示当前选中的选项 .....84
  - 3.10.4 下拉菜单核心组件  
PopupMenu .....85
  - 3.10.5 制作下拉菜单的注意事项 .....87





## 第4章 UI动画 .....88

- 4.1 常见的两种 UI 动画介绍 .....88
  - 4.1.1 要区分 UI 动画和 UI 特效两个概念 .....88
  - 4.1.2 关于 Tween 动画 .....88
  - 4.1.3 关于 Animation 动画 .....89
- 4.2 渐隐渐现动画（透明度动画） .....89
  - 4.2.1 透明度动画的介绍和应用 .....89
  - 4.2.2 使用透明度动画 TweenAlpha .....90
  - 4.2.3 使用透明度动画的注意事项 .....94
- 4.3 颜色变化动画（变色动画） .....95
  - 4.3.1 变色动画的介绍和应用 .....95
  - 4.3.2 使用颜色动画 TweenColor .....96
  - 4.3.3 使用颜色动画的注意事项 .....96
- 4.4 位置变换动画（位移动画） .....97
  - 4.4.1 位移动画的介绍和应用 .....97
  - 4.4.2 使用位移动画 TweenPosition .....98
  - 4.4.3 使用位移动画的注意事项 .....98
- 4.5 旋转变换动画（旋转动画） .....99
  - 4.5.1 旋转动画的介绍和应用 .....99
  - 4.5.2 使用旋转动画 TweenRotation .....99
  - 4.5.3 使用旋转动画的注意事项 .....100
- 4.6 大小变化动画（放缩动画） .....100
  - 4.6.1 放缩动画的介绍和应用 .....100
  - 4.6.2 使用放缩动画 TweenScale .....101

- 4.6.3 使用放缩动画的注意事项 .....101
- 4.7 Tween 动画总结 .....102
- 4.8 动画控制组件 UIPlayTween .....102
  - 4.8.1 为什么要用 UIPlayTween .....102
  - 4.8.2 动画核心组件 UIPlayTween 讲解 .....103
  - 4.8.3 使用 UIPlayTween 的注意事项 .....106
- 4.9 动画控制组件 UIPlayAnimation .....107
  - 4.9.1 为什么要用 UIPlayAnimation .....107
  - 4.9.2 为 UI 添加 Animation 组件 .....107
  - 4.9.3 动画核心组件 UIPlayAnimation 讲解 .....108
  - 4.9.4 使用 UIPlayAnimation 的注意事项 .....110



## 第5章 其他组件 .....111

- 5.1 使用 Toggle 制作页签 .....111
  - 5.1.1 页签的工作原理 .....111
  - 5.1.2 一个完整的页签界面 .....111
  - 5.1.3 制作两个页签按钮 .....111

5.1.4	使用 ToggleObjects 来记录 页签内容 .....	114
5.1.5	制作页签注意事项 .....	115
5.2	拖动摄像机来浏览超大界面 .....	115
5.2.1	拖动相机功能的介绍 和应用 .....	115
5.2.2	核心原理和组件介绍 .....	117
5.2.3	拖动相机浏览超大界面 的注意事项 .....	119
5.3	使用 Grid 自动排列 UI .....	120
5.3.1	自动排列 UI 的应用 .....	120
5.3.2	自动排列 UI 核心组件 Grid 介绍 .....	120
5.4	使用 DragObject 直接拖动物体 .....	122
5.5	让玩家通过拖动自由改变 控件大小 .....	124
5.6	制作序列帧精灵动画 (SpriteAnimation) .....	125
5.6.1	什么是序列帧精灵动画 .....	125
5.6.2	SpriteAnimation 组件 .....	125



## 第6章 NGUI 实战进阶 .....

6.1	UI 开发核心问题——UI 随屏幕 自适应 .....	127
6.1.1	屏幕分辨率对 UI 适配 的影响 .....	127
6.1.2	主流设备的屏幕分辨率 .....	129
6.1.3	自适应核心组件 Anchor 的使用 .....	130
6.1.4	使用 Anchor 的注意事项 .....	133
6.1.5	正式开发 UI 之前必须明确	

	的几个问题 .....	134
6.2	UI 元素的相对自适应 .....	134
6.2.1	什么是 UI 元素的相对 自适应 .....	134
6.2.2	Anchors 的介绍及使用 .....	134
6.2.3	使用 Anchors 的范例: 背景图的全屏适配 .....	137
6.2.4	使用 Anchors 的注意事项 .....	138
6.3	多摄像机同时协作运行 .....	139
6.3.1	摄像的渲染层的概念 .....	139
6.3.2	多摄像机协作的应用范围 .....	140
6.3.3	如何创建多个 UI 摄像机 .....	140
6.3.4	多摄像机协作的注意事项 .....	142
6.4	巧用九宫格以减少 UI 资源量 .....	142
6.4.1	项目安装包大小对项目 的影响 .....	142
6.4.2	UI 资源量对资源包大小和 内存的影响 .....	143
6.4.3	什么是九宫格 UI .....	143
6.4.4	如何让美术提供合适的 九宫格 UI 资源 .....	144
6.4.5	如何在 NGUI 中划分 九宫格 .....	144
6.4.6	如何使用九宫格 UI .....	147
6.4.7	去掉 Mipmap 以进一步降低 资源包大小和内存占用 .....	148
6.5	实战开发中 UI 资源制作标准 .....	148
6.5.1	为什么要设定 UI 资源 制作标准 .....	148
6.5.2	资源制作标准设定建议 .....	149
6.5.3	程序如何保证 UI 资源的 分辨率不失真 .....	150
6.5.4	针对各大平台设置单独的 尺寸和格式 .....	150
6.6	UI 事件监听的击穿 .....	151
6.6.1	什么是 UI 事件监听 的击穿 .....	151



6.6.2	如何避免和解决 UI 事件 监听的击穿	152
6.6.3	事件监听遮挡的妙用	153
6.7	开发之前的思考——UI 结构 设计	153
6.7.1	什么是 UI 结构设计	153
6.7.2	UI 结构设计遵循的一些 要点	153
6.7.3	需要的时候, 分场景以减轻 内存负担	154



## 第 7 章 用代码深度控制 UI 155

7.1	代码操作 NGUI 的原理	155
7.1.1	物体与组件的概念	155
7.1.2	怎样用代码操作 NGUI	155
7.1.3	获取组件的几种方法	158
7.1.4	迅速判断可以修改 的成员	160
7.2	动态加载 UI 元素	161
7.2.1	为什么游戏中会用到动态 加载 UI 元素	161

7.2.2	擅用 UI 元素的 Prefab	161
7.2.3	将一个物体设置为另一个物 体的子物体——NGUITools. AddChild()方法	162
7.2.4	NGUITools.AddChild()和 Instantiate 的区别	163
7.3	擅用 EventDelegate 事件委托	164
7.3.1	什么是 EventDelegate 事件委托	164
7.3.2	事件委托的用法	164
7.3.3	哪些地方可以使用 事件委托	166
7.4	巧用 EventTrigger 组件	167
7.4.1	什么是 EventTrigger 组件	167
7.4.2	EventTrigger 用法	168
7.5	常用组件的功能调用	168
7.5.1	UILabel	168
7.5.2	UISprite	169
7.5.3	UITexture	170
7.5.4	UIButton	170
7.5.5	UGrid	171
7.5.6	UISlider	171
7.5.7	UIToggle	172
7.5.8	UIInput	172
7.5.9	UIPanel	173
7.5.10	UICamera	173
7.6	动画的控制	174
7.6.1	为什么要把动画单独 提取出来	174
7.6.2	控制 Tween 动画	174
7.6.3	关于 PlayTween 和 PlayAnimation	176



## 第 8 章 实用案例演示.....177

- 8.1 角色头像状态栏制作.....177
  - 8.1.1 示意图和需求分析.....177
  - 8.1.2 设计并制作 UI.....178
  - 8.1.3 设计并编写代码.....181
- 8.2 场景加载的进度条界面制作.....187
  - 8.2.1 为什么要做这个界面.....187
  - 8.2.2 异步加载的概念.....187
  - 8.2.3 制作一个单独的加载  
界面场景.....188
  - 8.2.4 设计并编写代码.....189
- 8.3 技能快捷栏的制作.....194
  - 8.3.1 示意图和需求分析.....194
  - 8.3.2 设计并制作 UI.....194
  - 8.3.3 设计并编写代码.....197
- 8.4 角色头顶血条的跟随.....202
  - 8.4.1 角色头顶血条的跟随分析.....202
  - 8.4.2 制作血条的 UI.....202
  - 8.4.3 设计并编写代码.....203
- 8.5 NGUI 多语言切换的实现.....206
  - 8.5.1 什么是本地化.....206

- 8.5.2 NGUI 本地化的原理.....206
- 8.5.3 本地化案例演示.....210



## 第 9 章 常见疑难问题解答.....214

- 9.1 关于 NGUI 版本问题.....214
- 9.2 导入 NGUI 资源包出错.....214
- 9.3 如何创建两个 UIRoot.....215
- 9.4 如何让粒子在界面上正确显示.....215
- 9.5 为什么在父物体上增加透明度  
动画, 子物体没有跟着变化.....216
- 9.6 为什么动画播放一遍之后无法  
再次正常播放.....216
- 9.7 为什么 3DUI 模式下, UI 资源  
的尺寸 Snap 后和屏幕的大小比  
例不一致.....216
- 9.8 为什么 UI 不受灯光影响.....217
- 9.9 为什么 3D 模型放到 UIRoot 下  
就变得看不见了.....217
- 9.10 为什么 UI 单击后无法  
播放音效.....217
- 9.11 为什么 Depth 更大的图片反而  
被 Depth 小的图片遮住.....218
- 9.12 怎样判断点中的东西是 UI.....218
- 9.13 为什么 Label 的文字始终  
不够清晰、明亮.....218
- 9.14 为什么创建的物体有 BoxCollider  
却无法接收事件.....219
- 9.15 为什么改变了控件的父物体,  
导致了显示层级错乱.....220
- 9.16 关于 ScrollView 滑动的问题.....220

# 第1章 初识 NGUI

## 1.1 游戏 UI 开发介绍

### 1.1.1 什么是游戏 UI

UI 全称是 User Interface, 即用户界面。UI 的概念运用于各行各业, 例如电脑操作系统、手机、网站等, 几乎所有需要用户自主操作的地方都会涉及用户界面。UI 的职责是负责人机之间的交互, 它需要将关键信息、操作逻辑等展示给用户。好的 UI 设计不但能使操作变得易于理解、简单易用, 而且能做到简洁美观, 给用户带来舒适的操作体验。在软件设计中, UI 的设计是核心设计工作之一。

游戏是一种非常典型的互动娱乐, 不论在什么游戏中, 玩家都需要进行自己的操作, 而且频率非常高。在一些大型游戏中甚至会出现更复杂而频繁的操作, 例如竞技游戏和大型网络游戏等。这些在游戏中进行的操作, 让玩家体会到了游戏的乐趣, 而这一切都依赖于一整套游戏的 UI 机制。

在游戏中, UI 几乎无处不在, 例如进入游戏的菜单、输入账号密码的登录界面、创建和选择角色、角色血条状态栏和技能栏、场景雷达图、各种各样的提示框等。

### 1.1.2 UI 为何如此重要

在游戏中 UI 主要承担了以下职责:

#### 1. 游戏美术风格的重要组成部分

一个游戏由不同种类的美术资源组成, 例如角色、场景、特效、UI 等。每一个游戏都必须有一个统一的美术风格, 否则游戏将变成不伦不类的四不像。例如中国仙侠风格的游戏, 不



## 第1章 初识 NGUI

应该出现欧美魔幻的元素；同理，欧美魔幻风的游戏里，不应该出现中国仙侠的元素。UI 因其无处不在的特性，成为游戏设定美术风格时一个非常重要的考量指标。

### 2. 承担着重要的美观职责

当玩家打开游戏看到第一个游戏画面时，UI 就将一直陪伴着玩家直到玩家退出游戏。甚至大部分 UI 都是长久性地出现在玩家的游戏窗口中，例如玩家角色的状态栏、技能栏等。所以，UI 的美观和耐看，将会直接影响着玩家对这款游戏的美观程度的评价。

### 3. 负责清晰明了地展现游戏的操作方式

每一个游戏都有着自己的操作方式，越是大型游戏，操作逻辑往往会越复杂和庞大。好的 UI 设计，能让用户不依赖指引就能够迅速地知道自己应该怎样操作来进行游戏。

### 4. 能让玩家舒服、方便地进行人机交互

游戏 UI 的核心目的就是让玩家能进行自主操作，而游戏的机制往往很复杂，这个时候“人性化”就变得很重要。例如技能冷却完成之后，技能栏闪烁一下，玩家在游戏中不自觉地就能知道技能已经冷却完成。

#### 1.1.3 UI 开发的流程

在项目开发中，一般都是由策划人员、程序人员、美术人员、测试人员组成多个组协同进行开发。在开发 UI 时，首先应当由美术核心人员确立 UI 的整体美术风格标准，然后再进行 UI 的各个模块的开发。

在开发 UI 的模块时，首先应当由相关策划人员提出功能的需求，指出该模块的 UI 应该具备什么功能、能让玩家进行哪些操作逻辑。然后，由策划人员给出 UI 的控件布局图或者由美术人员自行设计布局图。再由 UI 美术人员进行 UI 的资源制作，制作完成后，将 UI 资源提交给客户端程序员。客户端程序员拿着美术人员提供的 UI 资源，按照 UI 的布局图和功能文档，最终完成该模块的功能开发。

#### 1.1.4 UI 开发的难点

UI 在开发中往往具备以下几个问题。

- (1) 需求不清晰。
- (2) 功能难实现。
- (3) 工作量很大。



(4) 优化不够好。

(5) 改动太频繁。

这些问题都是项目实际开发中非常常见的问题,特别是客户端程序员经常为以上五个问题伤脑筋。要规避它们,除了良好的沟通以外,还需要足够的对 UI 的思考,以完成一个良好的 UI 架构来提高抗风险指数,并扩充自身的知识面,以做到提早考虑到更多的特殊情况。

看完本书后,相信你一定能找到应对以上项目开发实际问题的最优方案!

## 1.2 什么是 NGUI

### 1.2.1 NGUI 插件介绍

NGUI 是专门针对 Unity 引擎、用 C# 语言编写的一套插件,经历了数十个版本的更迭之后,它已经成为了目前世界上应用最广、最成熟的 Unity 制作 UI 的插件,完美地弥补了 Unity 引擎原生 GUI 系统和 NewGUI 系统的各种不足之处。程序员可以利用它提供的一整套 UI 框架和事件通知系统来进行项目的 UI 设计和制作,NGUI 凭借其强大的功能、良好的优化和易用易学性,让大多数程序员都赞赏有加!

### 1.2.2 NGUI 的强大优势

#### 1. 成熟稳定

NGUI 经历了数十个版本的更迭,发展到现在几乎没有 BUG,并且完美支持跨平台和自适应。在这一点上,它远远超越了其他的 UI 插件。论成熟稳定,它比 Unity 的 NewGUI 还要更胜一筹。

#### 2. 功能丰富

NGUI 除了满足普遍的 UI 制作功能以外,还集成了大量的封装好的实用功能,比如拖曳、更多的事件监听、各种 Tween 动画、本地化等,甚至支持光照、法线、折射等特性。这是 NGUI 远远超过其他 UI 制作方式的地方,也是 NGUI 经历数十个版本更迭的积累。

#### 3. 操作方便

NGUI 的操作几乎都集中于 Inspector 面板中,并且不需要 Play 运行就能看到 UI 的结果。各个模块和组件封装非常好,需要功能时只要为其附上相应的 NGUI 组件就可以完成,大部分功能都不需要自己写代码。

## 第 1 章 初识 NGUI

---

### 4. 极致优化

目前最新的 NGUI 版本中，对于 UI 的渲染性能已经优化到了极致，使用 1 个 DrawCall 就能完成绝大部分 UI 的渲染。

### 5. 高灵活性

NGUI 都是以组件形式使用，程序员可以不借助任何外部的资源进行 UI 的制作，可以让任何一个控件通过改变其组件的方式，自由地变换为按钮、精灵、进度条、输入框等。

## 第2章 NGUI 基础

### 2.1 导入 NGUI 插件

#### 2.1.1 NGUI 版本介绍

NGUI 插件目前较新的版本是 3.6 以后的版本。

在 NGUI 3.0 以前的时期，底层的事件通信体系完全依赖于 `SendMessage`，这是一个效率比较底下的发消息方式，那个时期大多数 Unity 的开发者的使用当时很流行的 NGUI 2.6 版本，甚至目前还有少数开发者在使用。

在 NGUI 3.0 及以后的版本中，NGUI 进行了大革新，其中革命性的就是将整个的底层消息机制全部换为效率高的 `EventDelegate`。并且开始重视 NGUI 的性能优化，一直到 3.6 时期，NGUI 相比以前，提高了事件分发的效率，将性能优化到了极致，整合了大量的相近功能，并大大丰富了 NGUI 的功能类型。

在本书中，将使用比较新的 NGUI 3.6.8 版本进行讲解，书中可能部分截图并不是 3.6.8 的版本，但是 NGUI 3.6.0 及以后的版本几乎都一样，本书的知识点通用于 NGUI 3.6.0 及以后的所有版本，读者大可放心。

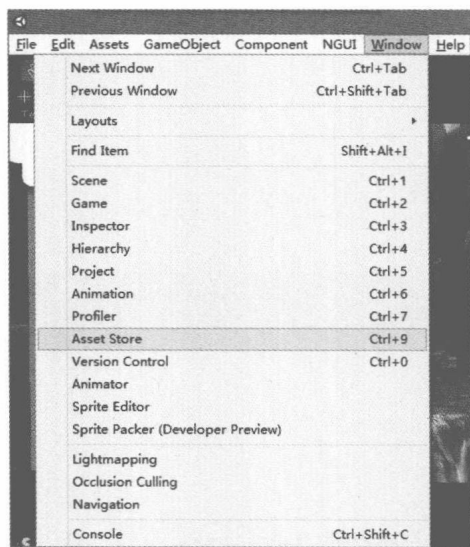
#### 2.1.2 NGUI 的下载和购买

NGUI 本身是一个付费插件，开发者可以从 Unity 官方的 AssetStore（官方提供的 Unity 资源买卖平台，里面有很多第三方的资源和插件出售，有的收费有的免费）中去购买，售价大约 95 美金（折合人民币 591 元）。用户可以从 Unity 引擎的编辑器界面的顶部 Window 菜单中选择 Asset Store 进入，如图 2.1 所示。也可以在浏览器中输入网址 <https://www.assetstore.unity3d.com/> 进入。

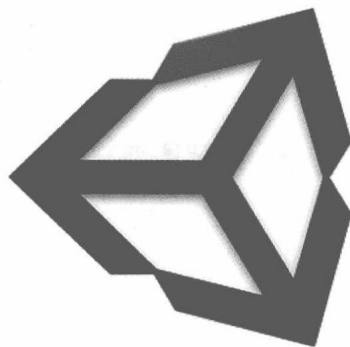


## 第2章 NGUI 基础

因为 NGUI 正版插件价格不菲,如果仅仅是为了学习和练习,开发者可以从网上去下载他人购买的 NGUI 插件包来使用,效果是一样的。不管是从官方购买的还是自己从第三方网站上下载的 NGUI 插件,它都应该是一个格式为“.unitypackage”的 Unity 资源包文件,如图 2.2 所示。



▲图 2.1



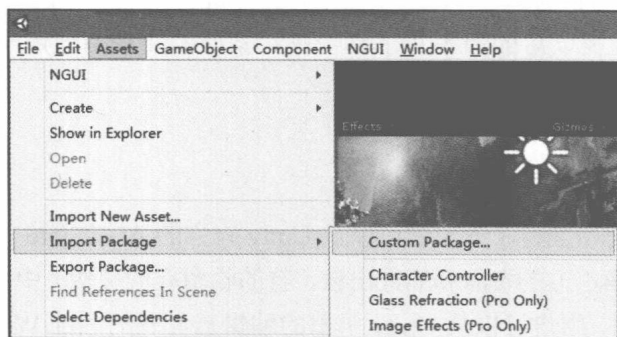
NGUI Next-Gen UI v3.6.8.unitypackage

▲图 2.2

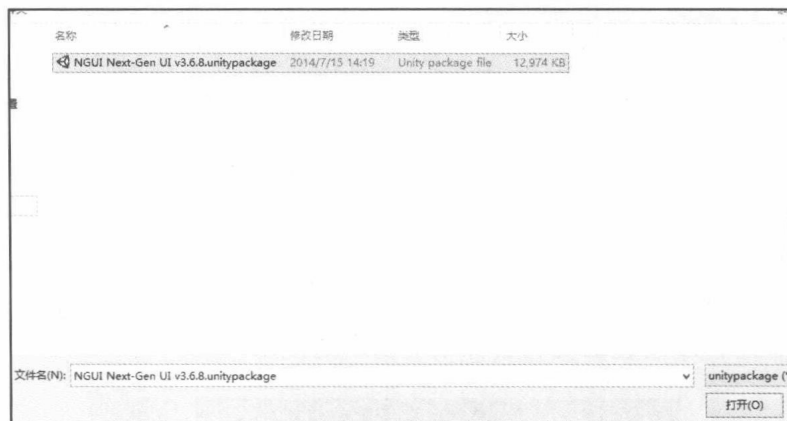
### 2.1.3 导入 NGUI 插件应用

当下载好 NGUI 的插件资源包之后,下面要做的就是将 NGUI 资源包导入到引擎中进行使用。

如图 2.3 所示,在 Unity 编辑器顶部菜单栏中的 Assets 菜单中选中 Import Package,然后选择 Custom Package (自定义资源包),弹出图 2.4 所示的资源路径窗口,在其中找到 NGUI 资源包所在的位置,单击“打开”按钮即可。



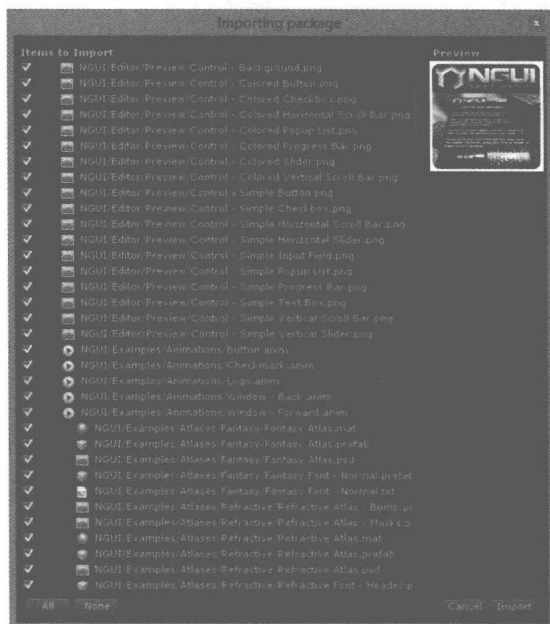
▲图 2.3



▲图 2.4

特别注意: 请将 NGUI 资源包放在一个没有中文的路径下再进行导入(例如 F:\Download\NGUI-Next-Gen-UI-v3.6.8)。因为 Unity 导入外部资源时, 将无法导入带有中文路径的资源, 例如 F:\下载\NGUI-Next-Gen-UI-v3.6.8 因为将 NGUI 放在了中文名文件夹“下载”之下, 将导致 NGUI 资源包无法成功导入。

单击“打开”按钮后, 等待 Unity 引擎解压资源包, 然后将会在 Unity 引擎界面中弹出图 2.5 所示的窗口, 展示该资源包的内容列表, 让用户选择导入哪些资源(默认情况下是全部选择), 此时因为其已经默认全部选择, 可以直接单击 Import 按钮, 将其全部导入。



▲图 2.5

## 第2章 NGUI 基础

导入成功后, 可以看到 NGUI 文件在 Project 视图中的结构图 (如图 2.6 所示), 其中 Editor 文件夹是编辑器所用的, 不用管它; Examples 文件夹是 Unity 制作的一些基本案例, 读者可以从 Examples 下面的 Scenes 文件夹中选择它制作的范例场景来参看一些基础功能的制作, 如图 2.7 所示。Resources 文件夹存储着 NGUI 自带范例所用到的资源。最后就是整个 NGUI 对于开发者来说最核心的文件夹: Scripts, 这里面是 NGUI 已经封装好的各个功能模块的脚本, 当需要使用时只要把相应的脚本变成 UI 物体的组件就可以进行相关的操作了, 本书在后文将会介绍更简单的使用方式。

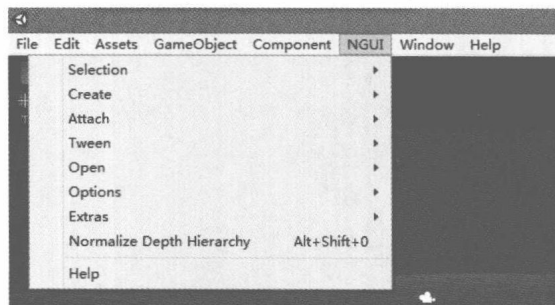


▲图 2.6



▲图 2.7

另外, 导入之后还会注意到 Unity 编辑器顶部菜单栏中多了一项 NGUI 菜单, 如图 2.8 所示, 这个菜单将会是以后使用 NGUI 制作 UI 系统最常用的一个菜单。



▲图 2.8



好了，到这里就已经说明你已经成功地导入了 NGUI 插件，这个插件包大约 12MB，不过完全不用担心它会让你的项目安装包增大很多，因为 NGUI 插件包导入后并不存在于 Resources 文件夹下面，所以在项目工程最后发布时，它只会将 NGUI 资源包中你所用到的部分纳入打包资源，对项目发布的游戏安装包体积的影响几乎可以忽略不计。

#### 2.1.4 导入常见问题

(1) 如果我的工程文件中已经导入过一次 NGUI 的资源包了，此时我再导入一个新的 NGUI 资源包会有什么结果？

答：会根据路径替换掉同名文件，并导入额外的新文件。

(2) 导入解压后，并没有弹出图 2.5 所示的资源包内容预览窗口，而是在 Unity 编辑器窗口底部报出了一行如图 2.9 所示的错误，怎么办？



▲图 2.9

答：这是因为你将 NGUI 资源包放在了一个带有中文名称的文件夹路径下，Unity 导入任何带有中文路径的资源包时，都会弹出这个错误导致无法导入。请将你要导入的资源包放在一个没有任何中文的路径下再进行导入。

(3) 我怎样查看我的 NGUI 版本是多少？

答：在 Project 窗口中，选择 Assets 文件夹下面的 NGUI 文件夹，然后会看到一个 ReadMe 的版本说明文件，这个文件名会带有版本号，如图 2.10 所示。



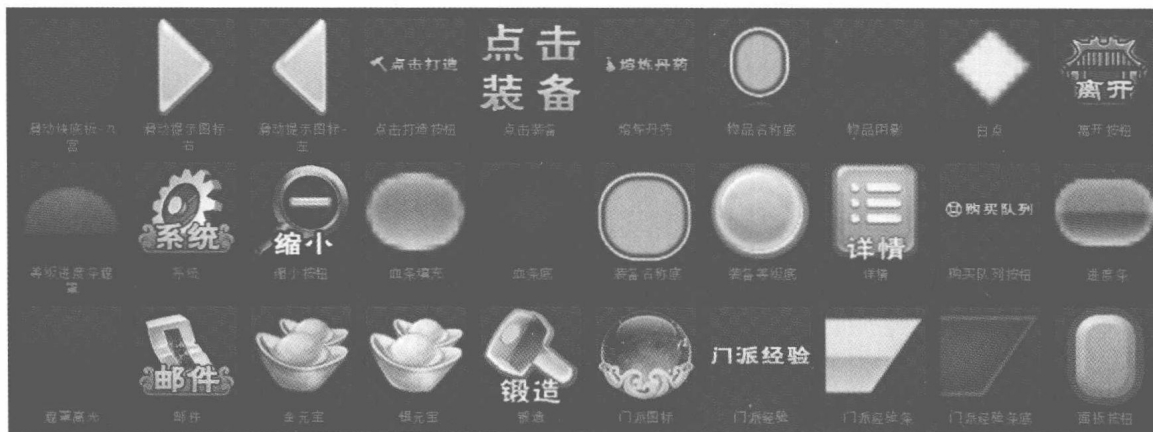
▲图 2.10

## 2.2 认识基本的 UI 资源

### 2.2.1 什么是 UI 精灵 (Sprite)

我们在制作 UI 时,经常将一些零碎的小的 UI 资源(比如,一个小箭头、一个按钮等)打包成一张大图,然后在使用时,只使用这个大图中的一部分(例如,只使用其中小箭头的那一小块),那么这一块“被切出来”的图片,就可以称之为精灵。

如图 2.11 所示的就是一个又一个的 UISprite。



▲图 2.11

### 2.2.2 什么是 UI 图集 (Atlas)

我们在制作 UI 时,会将一些零碎的小的 UI 资源打包到一张大图中,然后再通过精灵的方式对这张大图进行使用,这张大图就是一个图集。这样不但可以减小美术资源的总体积,还可以减少载入内存的操作(图集作为一张整图会被一次性载入到内存中)并提高渲染性能,而且还可以减少维护大量零碎小资源的麻烦。

如图 2.12 所示的则是一个由 Sprite 组成的图集。

### 2.2.3 什么是 UI 贴图 (Texture)

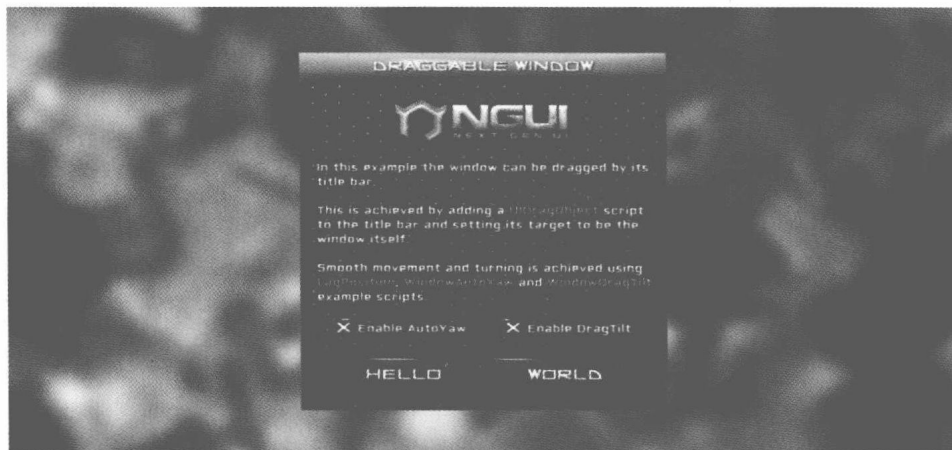
在 NGUI 中也有 UITexture 的概念,这个 UITexture 从功能用途上和 Sprite 精灵有很大的相似之处,都是为了显示一些图片资源。它和 Sprite 最大的区别在于,UITexture 是一张独立的图,不依托于任何的图集,这张 Texture 有自己的材质球和 Shader,每一个 UITexture 都将

消耗一个 DrawCall（不了解的读者可以理解为一个性能消耗单位）去渲染，每一个 UITexture 都将独立进行加载。

如图 2.13 中的大背景图就是 UITexture。



▲图 2.12



▲图 2.13

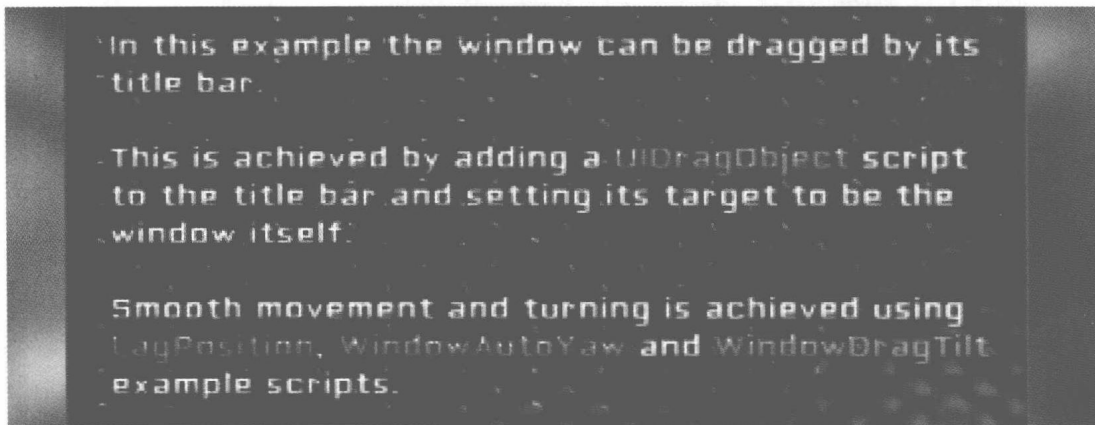


## 第 2 章 NGUI 基础

### 2.2.4 什么是 UI 标签 (Label)

标签 (Label) 在 NGUI 中并不是指一种标记物, 而是指一种纯文本的 UI 元素。凡是由程序在 UI 上打出来的字, 都属于标签的内容。例如, 如果你需要在界面上长期地显示一行字: 请打开背包进行整理, 那么这行字属于一个 Label。再比如, 如果你需要显示角色的生命值为 100/200, 这个数字会随着角色的生命值而变化, 这个生命值的数字也属于一个标签, 然后代码会根据角色的血量去读取并改变这个标签的内容。

如图 2.14 所示框中所有的文字信息都属于 Label。



▲图 2.14

### 2.2.5 什么是 UI 字体 (Font)

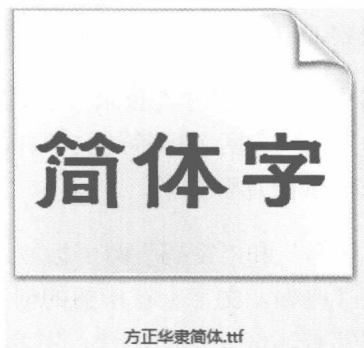
在制作 UI 的过程中, 不可能所有地方都由美术完成, 最典型的例子就是 UI 上面的文字。很多时候 UI 上面的文字都是不停地在进行变化, 并且没有什么复杂的艺术字效果, 不可能全部由美术制作成图片提供给程序, 这个时候就需要程序在 UI 上进行写字。程序在 UI 上写字时, 就将用到 UI 字体。

NGUI 的字体分为动态字体和静态字体。程序人员可以选择把某种特殊字体文件中的一些所需的字拿出来形成一张图, 然后打字时会从这张图里去调用文字 (类似于调用 Sprite), 这就是静态字体。也可以直接导入字体文件 (例如, 宋体、楷体等字体文件), 打字时只要字体文件里拥有的字都能正常使用, 这就是动态字体。当然, NGUI 有系统自带的默认动态字体。

如图 2.15 所示则为静态字体图集, 图 2.16 所示则为动态字体文件 (.ttf 格式)。



▲图 2.15

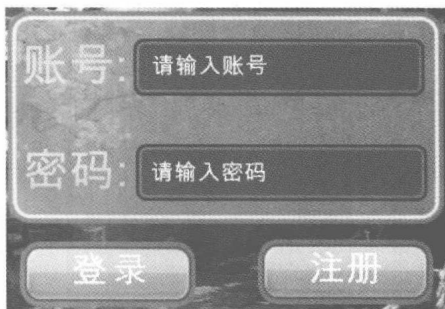


▲图 2.16

## 2.3 制作第一个 UI 图集

### 2.3.1 学会解剖 UI 的资源结构

为什么要剖析 UI 的资源结构？因为通常情况下，策划设计好 UI 的功能和大概布局之后，美术人员会做出一张 UI 的成品效果图，我们称之为 UI 设计图。然后，客户端程序需要根据自己的制作方式，告诉美术人员如何分割出相应的 UI 元素提交给程序，以完成制作。



▲图 2.17

下面以图 2.17 所示的 UI 设计图作为例子来讲解分析。首先说明一点，客户端程序一定要同时拿到 UI 设计图和 UI 功能描述文档才能彻底知道这个 UI 会进行什么样的操作逻辑，此时才能准确地对 UI 资源结构进行剖析。主要有以下一些方法。

- 看设计图，从相关设计文档了解该 UI 的功能。

在不了解 UI 功能的情况下，程序只看设计图不一定能准确地明白这个 UI 的全部用途，为了减少返工，请务必先了解该 UI 模块的全部功能。如图 2.17 所示，则是一个典型的登录 UI。

## 第2章 NGUI 基础

- 观察 UI 的设计图，判断哪些字是程序可以写的，哪些字是程序写不了的。

在图中我们发现“登录”“请输入账号”“请输入密码”等 Label。其中，“请输入账号”和“请输入密码”两个标签的文字并没有什么特殊的美术效果，完全可以由程序来完成，就不需要美术提供图片资源了。

而“账号”和“密码”两个标签，虽然本图中可以由程序完成，但是如果碰到那种有特殊美术效果（比如，文字上有华丽的渐变和高光等效果，程序是完成不了的）的，则需要让美术提供一张写有相应文字的图片，用 Sprite 去代替这个 Label。

后文我们会详细讲解什么情况下使用 Label。

- 通过设计图判断哪些是 Sprite，充分考虑复用性。

凡是零碎的、小的图片资源，都可以是 Sprite。如图 2.17 中，输入账号密码有一个底框，整个界面又有一个底框，这些都可以让美术人员切成单独的 Sprite，然后由程序来进行拼装。

在分割 Sprite 时，尽量分割得细一点，如图 2.17 中所示，我们可以将整个 UI 的大底框和输入账号密码的两个底框合并在一起切给程序，但是这样将导致这个 UI 图片无法用于其他地方。如果整个 UI 的大底框和输入账号密码的两个小底框分开来切，那么以后其他需要用底框的地方，就可以复用图中切出的 UI 资源。

相同的 UI 元素只需要一份就够了，例如图中输入账号的底框和输入密码的底框是一样的，于是只需要切出一份就可以了。

后文我们会详细讲解什么情况下使用 Sprite 和 Texture。

- 通过设计图判断出按钮资源的制作方式。

按钮笼统地分有两种形式，一种是普通按钮，也就是一张没有文字的按钮图片，程序人员在需要用时，就在上面写上“确定”等不同的、当前所需要的文字。另一种按钮则是图片按钮（以前旧版本 NGUI 的 ImageButton），这种按钮的特点是整个按钮就是一张图片，它既是按钮也是图片。

如图 2.17 中所示，登录和注册两个按钮几乎一模一样，只是上面的文字有区别，而“登录”和“注册”这几个文字明显是程序可以写出的文字效果，于是可以让美术人员切出一张普通按钮资源的底图，然后程序再在上面写字。

后文我们会详细讲解什么情况下使用按钮。



- 通过设计图判断其他控件。

要比较迅速地结合 UI 的功能判断出 UI 中的一些其他控件，例如假设这是一个人物生命法力的状态栏，那么一定要区分出 UI 中是否有用来显示人物生命法力值的进度条，如果有进度条，则进度条需要上面切一条表层条，下面切一个空槽底板条。

后文我们会对相关知识进行详细说明。

在剖析 UI 资源结构的时候，有无数种方案，每一种方案 UI 美术人员基本都能完成，程序人员也能完成相应的功能。但是，我们一定要通过充分的分析去使用一种相对更好的方案，这样对项目以后的修改和拓展都有好处。在剖析 UI 资源结构时一定要秉承以下几个原则：

- 尽量保证还原设计图的效果，不损失质量，这是前提。
- 尽量发现重复的元件，而且重复的元件只需要一份就足够。
- 尽量分割得零碎一点，避免多个元件合并在一起出图，这样对项目不利（后文会详细讲解）。
- 尽量使用九宫格来制作比较大的底板、底框等（后文会详细讲解）。
- UI 切图全部让美术人员以 PNG 格式导出。

### 2.3.2 如何导入切好的美术资源

当美术人员提交给你大量零碎的 UI 元件时，就可以开始进行 UI 的制作了。制作的首要任务是先将资源导入到引擎中去。首先在 Unity 的 Project 窗口下的 Assets 文件夹下面建立一个文件夹，将该文件夹命名为 Resources，表示这个项目的资源都放在这个文件夹下面，UI 资源也不例外。这个文件夹名字一定要设定为 Resources，不能改动。

这里讲一下为什么一定要设定一个名为 Resources 的专门的资源文件夹。Unity 开发中，如果涉及动态加载（在游戏中触发了某个条件才需要加载）的情况，比如需要临时生成一个烟火特效等，都会用到 Unity 的资源加载方法：Resources.Load（）；这要求需要被动态加载的资源一定要放在 Assets 下面一个叫 Resources 的文件夹中。而 UI 经常会涉及根据不同情况动态加载 UI 模块的情况，所以一般情况下，我们都会将 UI 的资源放在 Assets 目录下的 Resources 文件夹中。

但是也不是所有资源都放在 Resources 文件夹中就万事无忧了。因为 Unity 在生成游戏安装包时，对于 Resources 以外的文件夹，只会打包场景中用到的资源文件，而对于 Resources

## 第2章 NGUI 基础

文件夹, 因为涉及动态地往内存里加载资源, 所以 Unity 会无条件地全部打包。如果不加考量的把所有资源都放到 Resources 目录下, 可能会导致最终得到的游戏安装包体积变大。

这时我们可以在 Resources 目录下建一个名为 UI 的文件夹, 用来单独存储 UI 资源, 以避免 UI 资源和其他的角色、特效等资源放在一起难以管理。全部选中美术人员提交的 UI 元件, 一起拖曳到 Unity 界面中的 “Resources\UI” 目录下, 等待 Unity 读取资源之后, 我们导入的资源会显示在 “Resources\UI” 目录下, 这样就完成了资源的导入, 如图 2.18 所示。



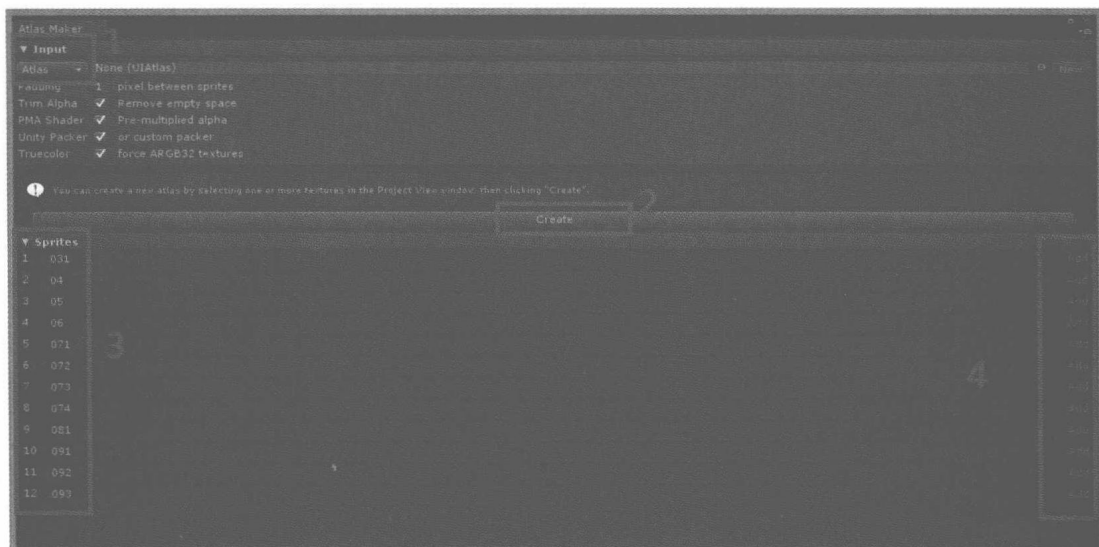
▲图 2.18

### 2.3.3 用 Atlas Maker 制作图集

刚刚我们已经将 UI 元件的源文件全部导入到了 Unity 中相应的文件夹目录下, 在 Unity 的 Project 窗口中全部选中刚才导入的 UI 元件, 单击鼠标右键, 选择最顶部 NGUI 菜单, 选择 Open Atlas Maker (Atlas Maker 是 NGUI 自带的一个 UI 图片打包工具), 这样就能自动将这些 UI 元件放入到 Atlas Maker 中, 如图 2.19 所示。

在 Mac 系统下, 右键菜单如果没有 NGUI 选项, 那么可以选中导入的 UI 元件, 在 Unity 顶部的菜单栏中, 选择 NGUI 菜单, 然后选择 Open\Atlas Maker 即可。

打开后的 Atlas Maker 界面如图 2.19 所示。



▲图 2.19

图 2.19 中, 标号为 1 的红框是已有图集的选择按钮, 如果你需要将这些新导入的 UI 素材资源全部新增到一个已有的图集里, 就可以单击这里。单击后能看到当前项目工程中已有的所有的图集, 然后可以选择其中一个图集, 此时标号为 2 的红框处的按钮将变成 Add/Update, 这样就可以新增或者更新这批资源到已有的选定的图集中了。

图 2.19 中标号为 2 的红框是主按钮, 当要打包的 UI 素材资源没有选定打包到某个已有图集中去时, 这个主按钮会显示 Create, 意为用这些资源创建一个全新的图集。如果通过标号 1 的红框处的按钮选择了一个已有图集的话, 这个主按钮将变成 Add/Update, 意为新增/更新当前这批 UI 资源到选中的图集中。更新的机制为同名的 Sprite 图片将会被替换。

图 2.19 中标号 3 处的红框显示的是当前选中的 UI 图片资源的序号和文件名称, 标号 4 处的红框显示的是这些资源哪些是新增的; 哪些是更新的; 哪些是已有的。在这里, 如果选中了一个已有图集, 那么该图集的 Sprite 也会一起显示出来, 如图 2.20 所示。在图 2.20 中, 红框 1 则表示当前选中了一个已有的名为 Fantasy Atlas 的图集。红框 2 是编辑图集和新建图集的按钮。红框 3 的按钮由 Create 变为了 Add/Update。红框 4 显示出了该图集已有的 Sprite, 在最尾部有一个删除按钮, 单击之后, 将会从这个图集中删除该 Sprite。如果此时单击 Add/Update 按钮, 那么 Sprites 列表中尾部标记为绿色的 Add 的精灵图片将会被新增到这个图集之中。

如果需要更新现有图集中的某一个精灵, 则将新的精灵图片文件的名称设为和它要替换的精灵的名称一样, 然后按照以上步骤选择它要替换的精灵所在的图集, 单击 Add/Update 即可实现直接替换资源。这在项目开发中是非常实用的, 当美术人员希望修改 UI 资源、用更新的 UI 资源替换之前的旧资源时, 这个自动更新功能将会让程序员非常方便。

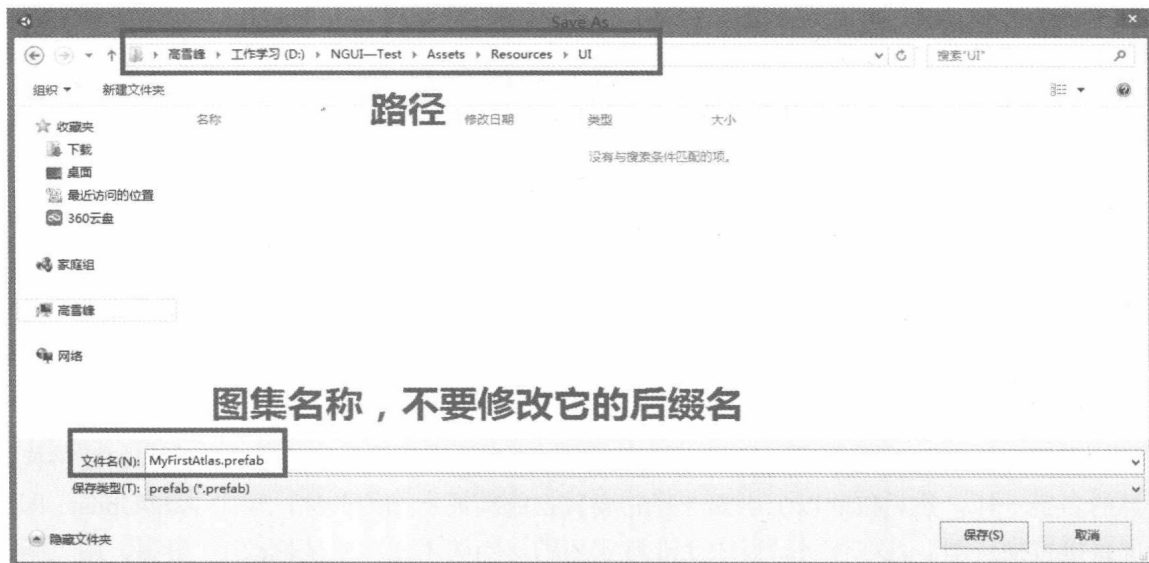


## 第2章 NGUI 基础



▲图 2.20

下面继续创建属于我们自己的第一个图集,当打开 Atlas Maker 之后,我们看到的是图 2.19 所示的界面。我们需要创建一个全新的图集,所以单击 Create 主按钮,然后会弹出 Save As 对话框,将路径定位到 Resources\UI 目录下,然后将图集的名称改为“MyFirstAtlas”,单击“保存”按钮即可,如图 2.21 所示。注意,不要改变文件的后缀名,文件保存后是一个 Prefab。



▲图 2.21

单击“保存”按钮之后，我们可以看到 Atlas Maker 界面已经变成了如图 2.22 所示的情况，这表明图集已经创建成功，中间的主按钮变成了 View Sprites，单击后可以预览该图集中所拥有的精灵。



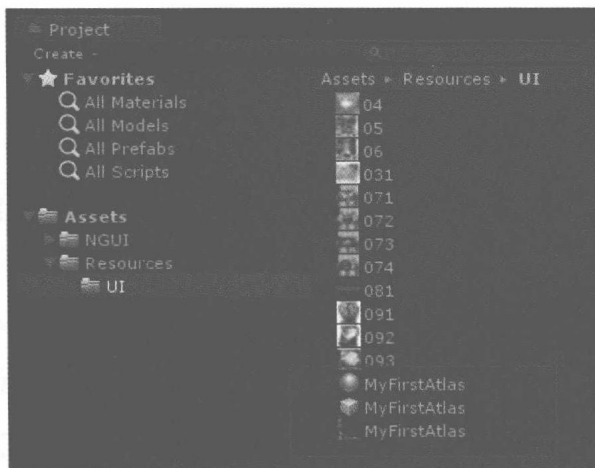
▲图 2.22

关闭 Atlas Maker 界面，然后注意看 Project 窗口中，Resources\UI 目录下除了我们之前导入的 UI 图片资源以外，多出了 3 个名为“MyFirstAtlas”的文件，如图 2.23 所示。这 3 个文件是一个图集必须具备的 3 个文件：图集的贴图、图集的材质球和图集的预设体。其中，第一个球形图标的 MyFirstAtlas 文件为图集的材质球；第二个蓝色方块的文件为图集的预设体；第三个图片缩略图文件则是图集的贴图，也就是精灵合成为一张整图之后的图片。

至此，我们的第一个图集就算制作完成了，这个图集在后面制作 UI 时就可以直接使用了。针对 Atlas，还有很多功能和用法，例如九宫格等，我们会在本书的后半部分讲到。

特别注明：在制作完 UI 图集之后，我们可以将之前导入 Unity 的 UI 资源源文件删除以减小资源量。

## 第2章 NGUI 基础



▲图 2.23

## 2.4 制作第一个 UI 字体

### 2.4.1 为什么要制作 UI 字体

在游戏的项目开发中,字体是经常会用到的东西,因为游戏中不论是聊天、公告、提示语还是界面显示,都会涉及用程序来写字。一般来说,会有系统默认字体供我们使用,但是出于以下两个原因我们经常会需要制作独特的字体。

- 系统字体的风格和美观程度等无法满足我们的需求。

一般来说,系统字体都比较死板、生硬,风格单一,经常无法满足项目需求。例如,我们希望游戏中所有文字都使用楷书来突出中国风,那么则需要我们自己植入楷书字体。再例如,我们需要在某些地方显示一些造型独特的字体,更需要制作我们自己独特的字体文件才能满足这种需求。

- 应对系统字体丢失的情况。

某些玩家(特别是安卓玩家)如果经常从网上下载一些个性化的字体管理软件,会很容易导致系统字体丢失,这种情况一旦发生,会导致游戏内所有的文字都不能正常显示。为了以防万一,我们需要植入一套自己的字体在游戏资源包内部。

### 2.4.2 静态字体和动态字体

我们在 2.2 节中已经介绍了什么是静态字体和什么是动态字体,这里我们来了解一下什么



情况下需要静态字体，什么情况下需要动态字体。

当有大面积的字体需求，并且需要的文字几乎涵盖大部分汉字时，我们就需要制作动态字体。与其说是制作动态字体，不如说是植入动态字体，因为在新版的 NGUI 中，制作动态字体只需要导入一个 TTF 格式的字体文件即可。

当在某些地方有特殊字体的需求，并且这种字体显示的文字有限时，例如受到伤害时，角色头顶需要飘出一个有艺术效果的数字来表示伤害量，这种字体效果只会显示 0~9 共 10 个数字而已，其他地方都用不到这种字体，那么这个时候我们就可以为它制作一个静态字体。

具体来说，静态字体和动态字体有以下实质区别。

- 静态字体中，如果需要用到的文字不多，打成图集后资源量往往比动态字体小，一个动态字体的 TTF 格式文件一般为 3~6MB。
- 静态字体可以通过提供一张自定义的含有所需文字的图片和一个配置文件（记录图片哪一块是哪个字的文件）来完成。动态字体只能通过导入整个 TTF 格式的字体文件完成。
- 静态字体中的字一般非常有限，只有极少数的字（否则图片资源会非常大），所以应用范围非常小，几乎很少用到静态字体。而动态字体几乎包含所有的文字，被广泛运用。

### 2.4.3 制作静态字体介绍

静态字体曾经风靡一时，原因是那时候 NGUI 旧版本对动态字体支持不是很好，所以很多时候得依赖静态字体。目前 NGUI 对动态字体支持很好，所以静态字体的应用就变得很少，只有在特殊情况下才使用。

要制作静态字体，需要将字筛选出来打成一个图集，并生成一份记录其中哪一块是哪个字的配置文件，这时可以借助一个名为 BMFont 的软件将其制作出来。制作出这两份文件之后，导入到 Unity 里。

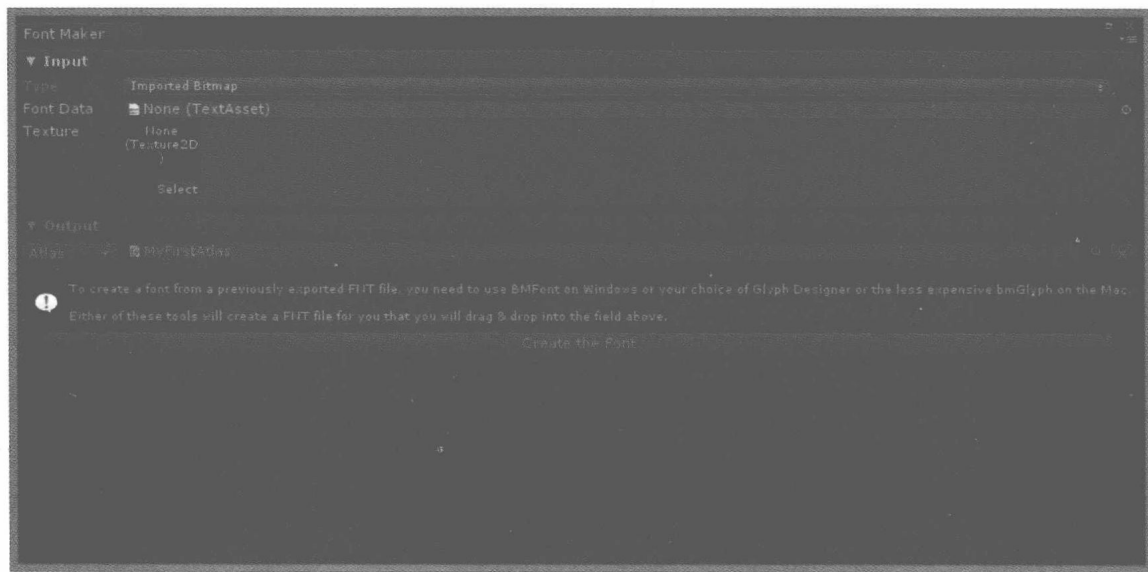
在 Unity 界面中，在 Project 窗口内单击鼠标右键，选择 NGUI 菜单，选择 Open Bitmap Font Maker，打开流程和打开 Atlas Maker 极其相似。Mac 电脑可以通过 Unity 顶部菜单栏中的 NGUI 菜单打开。

然后会弹出如图 2.24 所示的界面，在 Type 中选择 Imported Bitmap，然后在 Font Data 中选择我们之前制作出的那个记录文字位置信息的配置文件（最好是 TXT 格式），在 Texture 中选择我们之前制作出的那个文字图集，然后单击主按钮 Create the Font，即可创建出一个静态字体，创建出来的字体文件和制作图集得到的文件类似。制作好后，以后我们需要用字体的时

## 第2章 NGUI 基础

候, 选择这个字体的预设即可。

注意, 制作完成后不要删除导入的那张文字图集和文字位置的配置文件的源文件, 如果删除将会导致字体读不出文字。



▲图 2.24

### 2.4.4 制作动态字体介绍

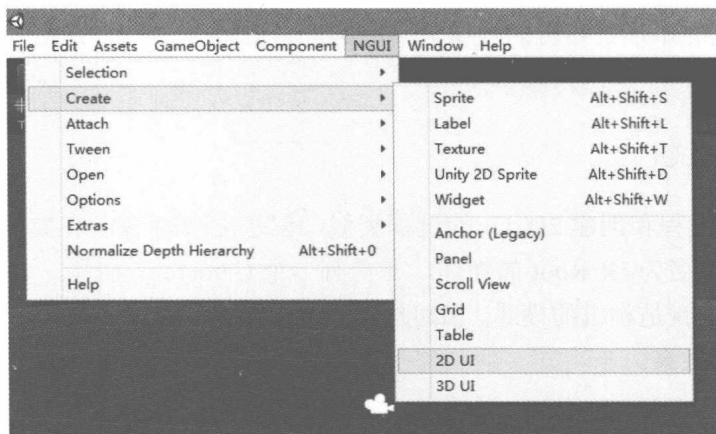
在新版本的 NGUI 中 (比如, 3.6 以后), 制作动态字体非常简单, 只需要从网上下载一个 TTF 格式的字体文件即可。然后将这个字体文件导入 Unity 中就算完成了, 以后需要用字体的时候, 就能直接调用这个字体。

注意, 字体文件要选择简体中文的, TTF 文件大小一般为 3~6MB, 如果远远超出了这个数, 一般来说很有可能是字体中包含了很多种语言。

## 2.5 创建第一个 UI

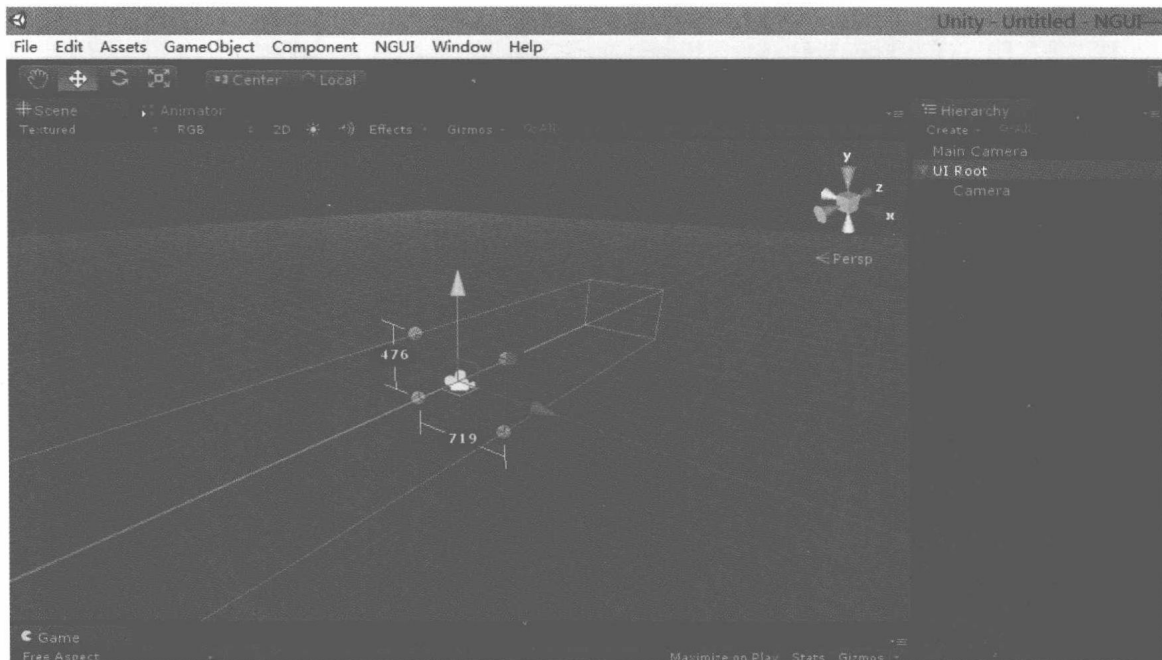
### 2.5.1 创建一个 2D UI

制作 UI 时, 首先我们要创建 UI 的“根”。在 Unity 顶部 NGUI 菜单中选择 **Create**, 然后选择 2D UI, 如图 2.25 所示。



▲图 2.25

创建完成后，我们能看到图 2.26 所示的景象，在 Scene 窗口中，NGUI 自动生成了一个名为 UI Root 的物体，其中带有一个 Camera 作为子物体。



▲图 2.26

这个新生成的 Camera，是 NGUI 生成的专门用来渲染 UI 的相机，当我们生成 NGUI 的 UI Root 时，就自动将生成的 UI 的 layer 设为了第 8 层。在这个相机中，只能看见第 8 层的物体，也就是只能看见 UI。因为是 2D UI，所以我们从图中可以看到相机是正交相机。



## 第2章 NGUI 基础

图 2.26 中红色的矩形是相机的视窗大小比例, 它会根据 Game 视图中的屏幕长宽比设置自动调整。

### 2.5.2 创建一个 3D UI

创建 3D UI 的过程和创建 2D UI 的过程类似, 创建出来的 3D UI 如图 2.27 所示, NGUI 依然自动生成了一个名为 UI Root 的物体, 并带有一个 Camera 子物体, 这个原理和 2D UI 类似, 其中最大的区别就是相机的模式。3D UI 的相机在 Scene 视窗中是一个正交摄像机, 将会支持 UI 的三维透视效果。

### 2.5.3 了解 UIRoot、UIPanel 和 UICamera 组件

在我们创建的 UI 中, 可以发现 UI Root 物体和 Camera 物体上面都带有 NGUI 特有的脚本组件, 其中 UI Root 物体上带有 UIRoot 和 UIPanel 两个组件, 而子物体 Camera 带有一个 UICamera 组件, 这几个组件都是 NGUI 体系中比较核心的组件, 下面我们来简单了解一下。



▲图 2.27

#### 1. UIRoot 组件

UIRoot 组件总是出现在 NGUI 的 UI “树” 的最顶层, 也就是那个 “根” 物体中。它的作

用是缩放 UI。我们在让美术人员作图时知道，UI 一般都是以像素作为单位，比如 1920\*1080 等，而 Unity 中则是以米为单位，如果一个 100\*100 像素的 UI 元件放入到一块 1000\*1000 分辨率的屏幕中，按理说这个 UI 元件应该是屏幕大小的 1%，但是因为 Unity 中的单位是米，所以它会从 100\*100 像素的大小变为 100\*100 米，会导致一个小 UI 元件变得非常之巨大。这个时候 UIRoot 会通过屏幕来缩放 UI 控件，让 UI 控件从视觉上是正常的。

在 UIRoot 组件中，它提供了 3 种缩放的方式，也就是 UIRoot 组件下的 Type 值。这 3 种方式分别为 PixelPerfect、FixedSize、FixedSizeOnMobiles。

PixelPerfect 是指永远保持像素大小不变，比如一张 100\*100 像素的图片，在 500\*500 分辨率的屏幕上，它是 100\*100 像素，在 1000\*1000 分辨率的屏幕上，它依然还是 100\*100 像素，因为它的源文件就是这个大小，而 PixelPerfect 让它一直保持这个大小。这样就可以让 UI 的图片永远是最清晰的，但是会导致分辨率高的屏幕下 UI 显得特别小；分辨率低的屏幕下 UI 显得特别大。

FixedSize 是和上一种缩放方案完全相反的方案。在 FixedSize 下，NGUI 将不再保护图片的原尺寸，只会关心 NGUI 自己所需要的缩放参数，这种模式下必须设置 UIRoot 的 ManualHeight 值，然后 NGUI 会将所有的控件按照和这个值的高度比例进行缩放。例如，设置 ManualHeight 为 1000，然后一张 100\*100 的图片在高度为 1000 的屏幕分辨率下占 1/10 的高度，那么当 UI 放到一个分辨率为 500\*500 的屏幕上时，它依然占 1/10 的高度，只不过图片尺寸被自动放缩为 50\*50，这样就保证了 UI 和屏幕分辨率的比例是一定的。

FixedSizeOnMobiles 是两种方案的结合体，它会让 UI 在 PC、Mac、Linux 系统下自动采用 PixelPerfect，而在移动设备上自动采用 FixedSize。

如果没有选择 FixedSize，那么必须设置另外两种缩放模式下的 MinimumHeight 和 MaximumHeight 两个值，代表最大高度和最小高度。例如选择 PixelPerfect 模式，将 MinimumHeight 设置为 720，将 MaximumHeight 设置为 900，那么在一个分辨率为 800\*600 的屏幕上，因为屏幕分辨率的高度小于 UIRoot 里的最小高度，UIRoot 就会按照 FixedSize 模式下 ManualHeight 为 720 的情况进行处理；同理，如果将 UI 放到一个分辨率为 1920\*1080 的屏幕上，因为该屏幕分辨率的高度 1080 大于设置的 900，于是 UIRoot 就会按照 FixedSize 模式下 ManualHeight 为 900 的情况进行处理。

值得注意的是，在 3.7.0 以后的 NGUI 上，UIRoot 的缩放模式改为了。

- Flexible，等同于上文讲到的 PixelPerfect。
- Constrained，等同于上文讲到的 FixedSize。

## 第2章 NGUI 基础

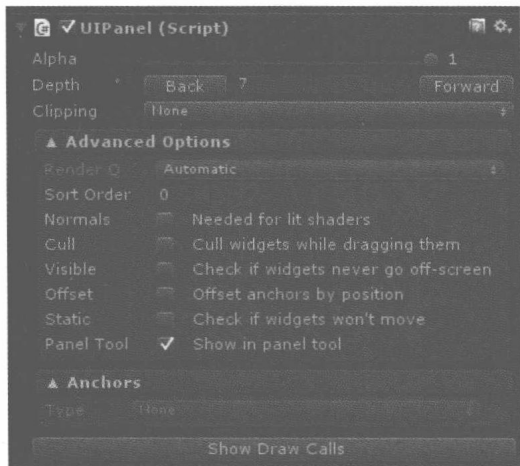
- ConstrainedOnMobiles, 等同于上文讲到的 FixedSizeOnMobiles。

功能上几乎完全一样。

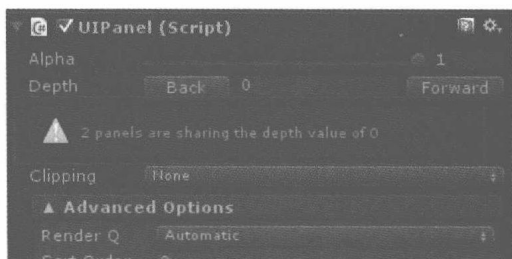
### 2. UIPanel 组件

如图 2.28 所示, UIPanel 有很多属性。其中, Alpha 属性顾名思义是透明度, 默认为 1 不透明。它将控制它旗下所有 Widget (所有的 UI 控件都将带有 Widget, 因为它们都继承自 Widget) 的透明度, 也就是它会让它的子物体里的所有 UI 控件都一起发生透明度变化, 可以用来做整个 UI 的淡入淡出以及隐藏等。

Depth 深度属性, 这是一个极其重要的属性。在 NGUI 中, 每一个 Panel 都有 Depth, 每一个 Widget 控件也有 Depth, Depth 将决定渲染的顺序, 直接影响了 UI 之间的前后重叠关系。Depth 越高的控件将会显示在视野的上层, Depth 越高的 Panel 也会显示在视野的上层。但是 Panel 的 Depth 权重远远高于 Widget, 也就是说, 在大部分情况下, 属于低 Depth 的 Panel 的控件, 不管这个控件本身的 Depth 为多少, 它都将显示在高 Depth 的 Panel 的控件后面。当你有多个 Panel 的时候, 例如你制作了很多面板界面, 每一个界面都有一个 Panel, 那么此时尽量保证这些 Panel 不要共用同一个 Depth, 因为这将导致 NGUI 在渲染时无法以 1 个 DrawCall 完成, 会以增加 DrawCall 的方式来保证渲染顺序不混乱, 这样就增大了性能的开销。不过 NGUI 在碰到 Panel 有共用 Depth 的情况时会做出提醒, 如图 2.29 红框部分所示。



▲图 2.28



▲图 2.29

Clipping 是剪辑视窗的功能, 它将可以让一个面板只显示某一块区域, 关于这部分知识后文我们再讲解。



在高级选项中, 我们讲解一些初学者需要了解的。Render Q 可以理解为渲染顺序, 默认为自动设置。这个选项在和粒子系统结合使用时会有影响, 我们后文会说明。如果该 Panel 下的 UI 需要被灯光影响到 (NGUI 的 UI 是默认不接收灯光照射效果的), 需要勾选 Normals。如果该 Panel 下面所有的 UI 控件都不会被移动, 那么可以勾选 Static 来将它们设置为静态的, 这样会导致该 Panel 下所有的控件都将忽略位置、旋转、缩放的操作, 永远保持不动。虽然这样可以提高一些性能, 但是慎重使用。

单击 Show Draw Calls 按钮, 可以看到该 Panel 下所有的 DrawCall 消耗情况, 如图 2.30 所示。

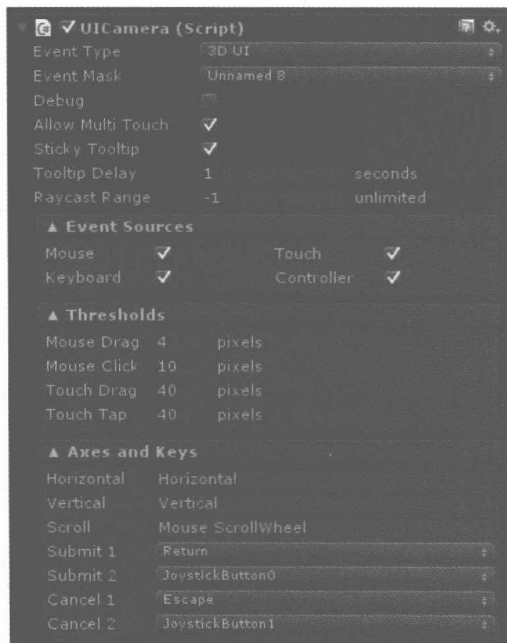


▲图 2.30

### 3. UICamera 组件

图 2.31 所示为 UICamera 组件的截图, UICamera 这个组件的核心作用是: 让带有这个组件的摄像机渲染出的物体能够接收 NGUI 的事件。如果我们自己创建了一个物体, 并且希望对这个物体使用一些 NGUI 中的事件, 例如 OnPress()、OnDrag()等, 就需要为渲染这个物体的摄像机添加 UICamera 组件。

## 第2章 NGUI 基础



▲图 2.31

在 UICamera 中,大部分设置我们都不需要去操心,它让我们的事件支持多点触摸、鼠标键盘触摸屏等事件的接收。但是要注意的是 EventMask 这个选项,这个 EventMask 和相机中的 CullingMask 非常相似,相机的 CullingMask 是为了选择渲染那些层的物体,这里的 EventMask 是为了选择接收那些层的事件。UICamera 会默认只接收我们创建 UI 时被自动设置的那个 layer,但是,如果我们在制作 UI 过程中,在创建 UI 后因为某些原因修改了 UI 的层,一定要将 UICamera 的 EventMask 修改过来,否则将会发现,我们单击 UI 没有反应,因为它接收不到这个 layer 的物体事件。

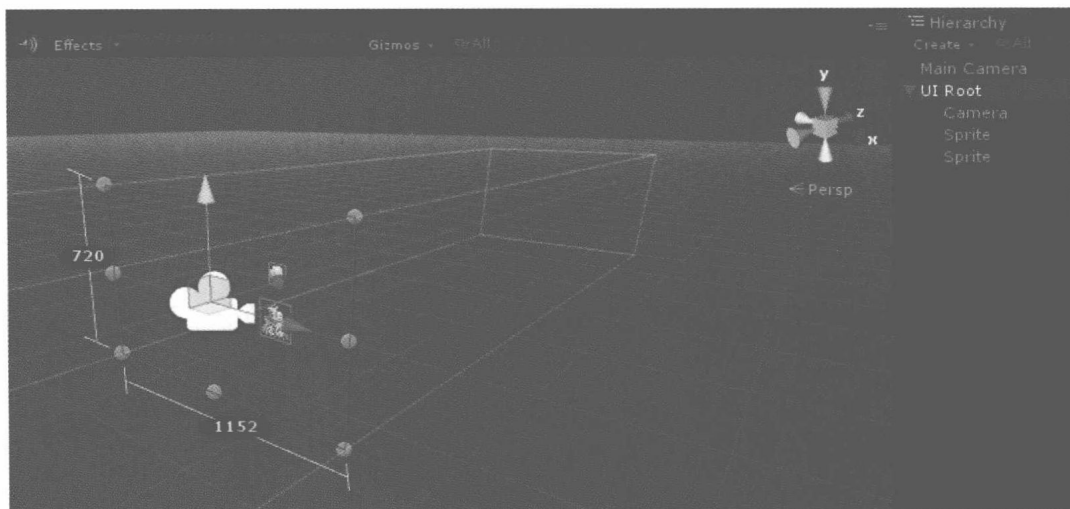
关于这 3 个最基础的控件讲了这么多,其中有很多都是较少用到,主要目的是加快对 NGUI 概念的形成。具体在需要的时候应该进行什么样的操作,我们后面的一些实战内容中会讲到。

## 2.6 2DUI 和 3DUI 的工作原理

### 2.6.1 2DUI 的工作原理

先创建一个 2DUI (创建方法上文已讲过),然后在 2DUI 的“根”UIRoot 下创建两个精灵(创建方法后面会详解)。然后得到的效果如图 2.32 所示。

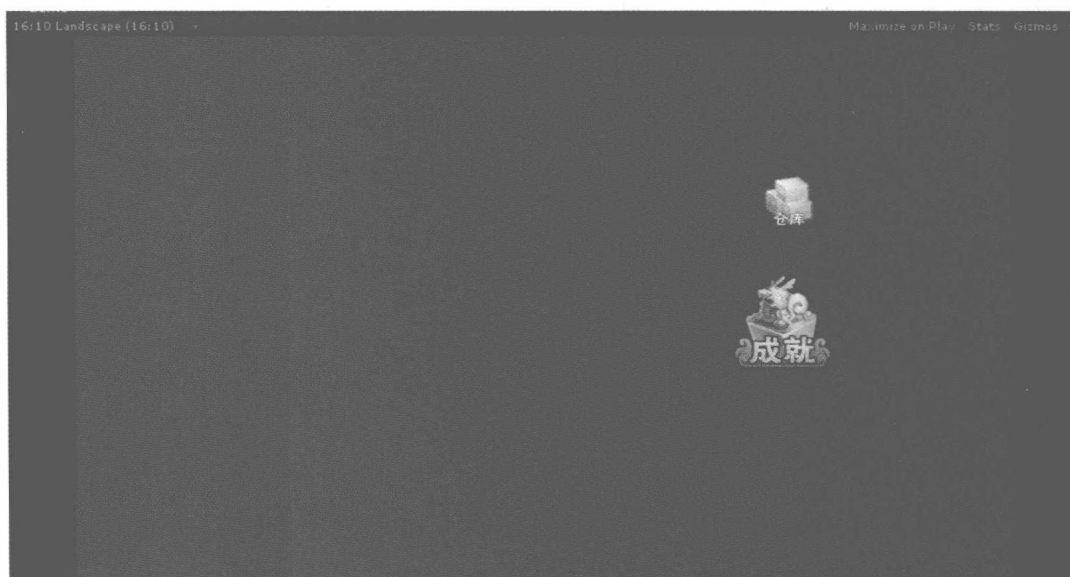
## 2.6 2DUI 和 3DUI 的工作原理



▲图 2.32

从图 2.32 中可以看到，创建的两个 **Sprite** 为两个按钮图片，它们的位置在 **UIRoot** 的红框（视窗）上，也就是 **Sprite** 的 **z** 轴、相机的 **z** 轴、**UIRoot** 的 **z** 轴都为 0，因为 2DUI 都是纯粹的 2D 图片按层次显示，不会出现三维立体效果，所以都是直接紧贴着视窗，只要 UI 控件在 **UIRoot** 的红框范围内，那么 UI 就能够正常显示在 **Game** 上。在 **Game** 视图中，我们看到的景象如图 2.33 所示。

2DUI 最本质的意义是：UI 摄像机是一个正交摄像机。



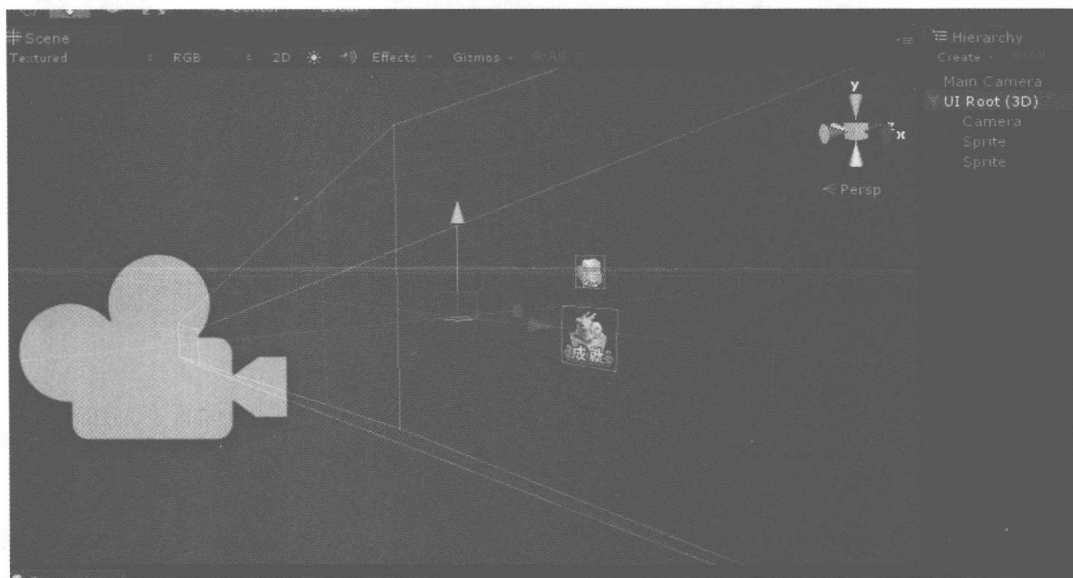
▲图 2.33



## 第2章 NGUI 基础

### 2.6.2 3DUI 的工作原理

同上 2.6.1 小节一样, 先创建一个 3DUI, 然后在“根”下面创建两个 Sprite, 得到如图 2.34 所示的景象。



▲图 2.34

从图 2.34 中可以看到, 在 3DUI 下, 创建的 UI 控件都在一个三维立体空间中, 摄像机是一个透视的摄像机, 这和 2DUI 有着截然不同的区别, 因为 2DUI 是一个正交摄像机。

3DUI 中 UIRoot 的坐标点如图 2.34 所示, 是在三维空间的一个点上, 这个位置是创建 UI 时自动定义好的, 以后创建的 UI 控件都会自动地在这个点所在的面上生成(自动统一到 UIRoot 的 z 轴, 因为 Sprite 属于 UIRoot 的子物体), 所以图 2.34 中 UIRoot 根物体和两个创建的 Sprite 的 z 轴都为 0, 而 Camera 的 z 轴为 -700。

我们在图 2.34 生成的两个 Sprite 用 3DUI 做出的效果, 在 Game 视图里看, 和 2DUI 几乎是一模一样的, 如图 2.35 所示。

但是需要注意的是, 如果需要将 2DUI 改为 3DUI, 不是简单地将摄像机改为透视模式就行了, 在 NGUI 3.6.0 以后的某些版本中, 这样会导致看不到任何 UI 控件(这些版本的 2DUI 的控件和 Camera、UIRoot 三者在同一个 z 轴面上, 而变成 3D 摄像机后是看不到和摄像机同 z 轴的物体), 如果将 3DUI 的摄像机直接改为正交模式, 也不能简单地变成 2DUI, 因为正交相机的 Size 并不和 NGUI 默认的值一样。所以, 在进行 UI 开发时, 最好先思考一下将要使用 2D 还是 3DUI 更好。



▲图 2.35

### 2.6.3 如何判断该选择哪一种 UI

上文中我们讲解了 2DUI 和 3DUI 的原理, 那么在实际的项目开发中, 如何来判断应该使用哪一种 UI 呢? 有以下一些规律可以参考。

- (1) 新版本的 NGUI 对 3DUI 支持很好, 如果 3DUI 和 2DUI 选择哪一个都行的情况下, 建议选择 3DUI, 扩展性更强。
- (2) 如果出现 UI 不允许有远近透视的大小变化, 必须选择 2DUI。例如, 角色头顶的血条, 在人物跑远了之后如果血条并不会变小, 这时血条就必须用 2DUI 做。
- (3) 如果要出现 UI 有三维变换的效果, 例如, 由远及近的变大、三维旋转等, 就必须用 3DUI。
- (4) 无法明确知道应该用哪一种 UI 的情况下, 建议选用 3DUI。
- (5) 不论用哪一种 UI, 其实本质上只是一个摄像机的区别, 基本上都能实现 UI 效果, 只是需要的处理不一样, 所以, 如果选择错了 UI 模式也不用太着急, 总有很多办法来解决的。

## 2.7 深度 (Depth) 概念

### 2.7.1 强化对深度的理解

深度的概念将会一直伴随着 UI 的制作过程, 是 UI 中一个非常重要的概念。我们在 2.5.3

## 第2章 NGUI 基础

小节中讲解 UIPanel 时已经讲解了深度的概念, 这里我们再强化一下对深度的理解。在老版本的 NGUI 中, UI 的显示层次关系是依靠  $z$  轴进行的。在新版本的 NGUI 中, 所有 UI 的  $z$  轴都被统一, 然后用深度来决定和管理显示的层次关系。关于深度, 我们要记住一下关键点。

(1) 每一个 UIPanel 和每一个 UI 控件都一定会有一个 Depth, 深度值大代表显示的优先级高 (会越趋向于在界面更上层显示)。

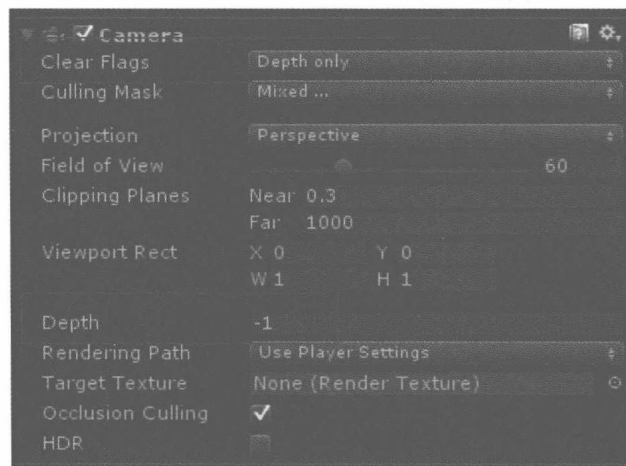
(2) Depth 决定的是 UI 的显示层级关系, 一个 UI 控件是否显示在最上层是由它所属的 Panel 的 Depth 和它本身的 Depth 决定的。一般情况下, 属于低 Depth 的 Panel 的控件, 不管这个控件本身的 Depth 为多少, 它都将显示在高 Depth 的 Panel 的控件后面 (被高 Depth 的 Panel 遮住)。

(3) 尽量不要让 Panel 之间共享同一个 Depth, 这样会导致性能消耗增加。

(4) 制作 Panel 和 UI 控件时, 记得考虑一下它所属的 panel 和它自身的 Depth 是否能让它显示在正确的层次关系上。

### 2.7.2 小心相机的深度

我们在场景中的每一个 Camera 也有一个渲染深度, 如图 2.36 所示。



▲图 2.36

在前文中我们学习到 NGUI 创建时, 都会创建一个它独有的相机。这个相机其实就是 Unity 中普通的 Camera, 然后为其附加了一个 UICamera 的组件。需要注意的是, 所有的 Camera 也都有一个 Depth, 这个 Depth 会影响到 UI 中的 Depth, 特别是场景有多个 Camera 来渲染不同层次的 UI 时, 这个影响会比较大。具体我们得遵循以下这些规律。



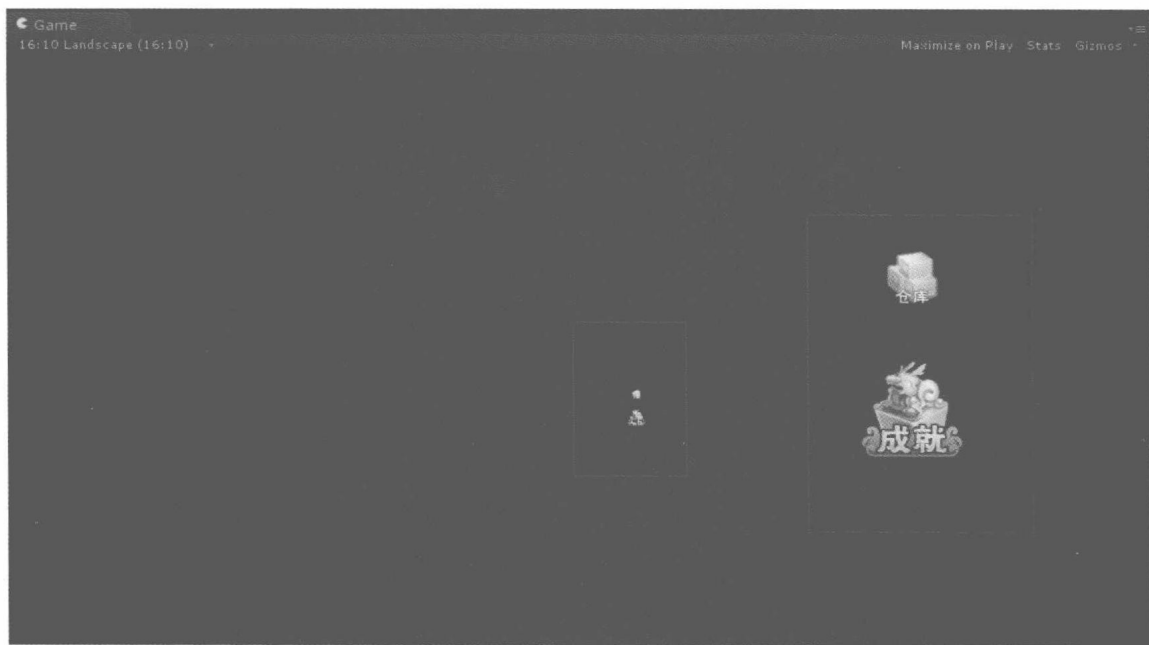
(1) 相机的 Depth 永远是最高级的, 也就是高 Depth 相机所看到的画面, 永远在低 Depth 相机所看到的画面之上。

(2) 如果需要相机有视觉穿透效果 (只渲染所看到的東西, 其他地方透掉显示其他相机所看到的画面), 需要将相机的 ClearFlags 设置为 DepthOnly。

(3) 并不是只有负责渲染 NGUI 的相机的 Depth 会有影响, 所有的相机 (比如默认存在的渲染场景的 MainCamera) 的 Depth 都受此规律影响。例如, 如果将照射 UI 的摄像机的深度设为 0, 然后将照射场景的相机深度设为 1, 那么, 将看到场景把所有的 UI 遮住。

(4) 创建 UI 时, UIRoot 下生成的相机默认 Depth 是比场景中的相机深度高的, 不过当场景内有多个摄像机时, 一定要管理好每个摄像机的 ClearFlags 和 Depth。

(5) 当场景内有多个摄像机时, 一定要检查摄像机的 CullingMask 不要渲染重复的 Layer, 否则可能导致显示双重画面。如图 2.37 所示, UI 画面被两个相机同时看到, 显示了两份 (因为两个相机所在的位置不一样, 所以看到的大小会不一样)。



▲图 2.37

## 第3章 核心组件

### 3.1 什么是 UI 控件

UI 控件这个词是我们制作 UI 过程中经常碰到的一个词，那么到底什么是 UI 控件呢？具体来说，UI 控件是指一个界面中可以操控的元件，如按钮、输入框等。但是在开发游戏的过程中，一般来说 UI 控件可以指界面中所有的 UI 元件，包括精灵、贴图、标签、输入框、下拉列表、按钮等，大家不需要对这个概念深究，在使用 NGUI 时，凡是带有 `Widget` 参数（后文要讲）的组件都是控件。

### 3.2 制作精灵 ( `UISprite` )

#### 3.2.1 怎样判断是否应该使用精灵

在一套 UI 中，精灵是一种非常常见的元件。当我们制作 UI 时，如果需要显示一张图片，需要先判断这个图片是否应该制作到图集里去，然后用精灵的方式去使用它，一般来说，可以遵循以下规律。

(1) 首先说明一点，精灵是一个很基础的 UI 元件，经常不会独立使用，很多其他控件都会用到精灵，比如一个进度条，需要用到一个表示空槽的精灵，和一个上面走进度的条子精灵。所以，精灵有的时候并不是独立使用的。

(2) 对于一些展示型的图片，不会变化，只是起一个展示作用，例如，一个界面上的一个小花纹装饰，如果它不大，它一般都是以精灵的方式去制作。

(3) 如果要显示一个图片，它形状不规则，长宽不是 2 的 N 次方，那么一定要使用精灵。

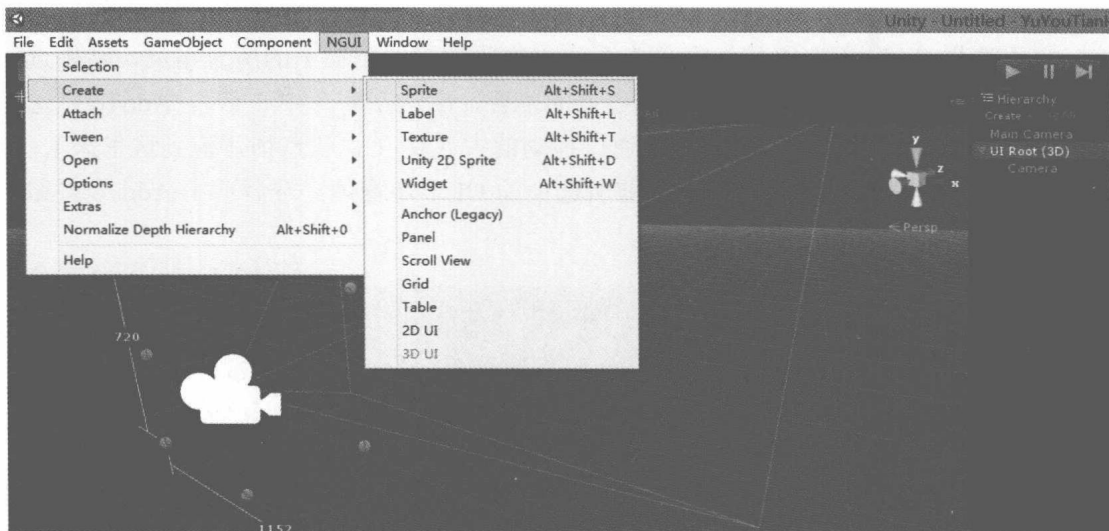
因为 Unity 对非 2 的 N 次方的图片处理要慢很多。

(4) 如果这个 UI 元件经常性地出现, 那么最好使用精灵, 因为, 这样它就可以和图集一起被载入内存, 并不用新增一个 DrawCall 去渲染它。

### 3.2.2 创建精灵

#### 1. 第一种创建方式

首先, 我们创建一个 3DUI (2DUI 也行, 这个没有任何影响), 然后选中 UIRoot, 单击 Unity 顶部菜单中的 NGUI 菜单, 选择 Create, 然后选择 Sprite, 如图 3.1 所示。这样就能在 UIRoot 下面自动创建出一个带 Sprite 组件的物体, 这就算创建成功了。



▲图 3.1

特别说明一下, NGUI 创建物体时会在你选中的那个 UI 物体 (可视为一个节点) 下进行创建, 如果你没有选中任何的 UI 节点, 它会默认在 UIRoot 下创建。创建出的 UI 控件的本地坐标都为 0 (相当于 Reset 了一下, 和父节点的位置保持一样), 所以, 使用 3DUI 的时候要注意, 不要在 UIRoot 所附带的 Camera 下面创建 UI 元件, 否则会导致 UI 和相机在一个位置, 导致相机看不到。

#### 2. 第二种创建方式

使用旧版本的创建方式, 在 Unity 顶部菜单中选择 NGUI 菜单, 选择 Open, 选择 WidgetWizard(Legacy), 如图 3.2 所示。

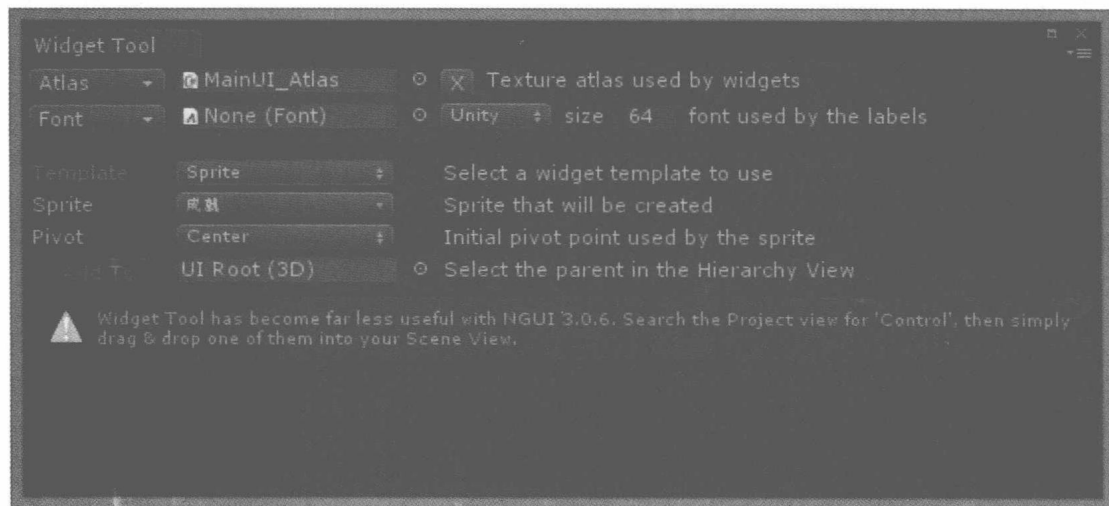


## 第3章 核心组件



▲图 3.2

打开后，会弹出如图 3.3 所示的界面，其中选择想要创建的精灵所在的 Atlas，然后在 Template 中选择 Sprite，在 Sprite 栏单击会弹出你所设置的图集中的所有精灵，从中选择你要创建的精灵，Pivot 是精灵的锚点（中心点的位置，默认在图片中心点）。AddTo 是选择你要在哪个 UI 节点下进行创建（可以通过拖动的方式将 UI 节点物体拖到这里来），这个 AddTo 的默认值是你打开这个菜单之前所选中的 UI 节点物体。然后单击 AddTo 按钮，即可完成创建。

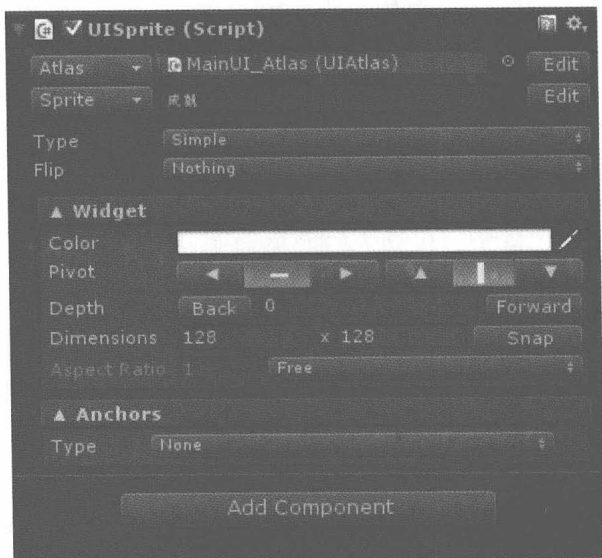


▲图 3.3

### 3. 第三种创建方式（不推荐）

这种方式是不用 NGUI 的菜单来创建，通过 Unity 的空物体，然后为其附加相应组件来自制 UI 控件。

首先在 Unity 顶部菜单中选择 **GameObject**, 然后选择 **CreatEmpty**, 这样就在场景中创建了一个空物体, 然后将它的名字改为 **Sprite** (这个物体的名字可自由定义), 再将这个空物体拖动到 **UIRoot** 下, 使它成为 **UIRoot** 下的一个子物体, 将这个空物体的 **transform** 组件 **Reset** 一下, 这样这个物体就和 **UIRoot** 根节点保持一样的位置了。然后将这个空物体的 **Layer** 改为和 **UIRoot** 的 **Layer** 一样, 否则 UI 摄像机将无法渲染它。在这个空物体的 **Inspector** 面板中, 单击 **Add Component** 按钮, 选择 **NGUI**, 选择 **UI**, 再选择 **NGUI Sprite**, 就为这个空物体附上了 **Sprite** 组件, 如图 3.4 所示。



▲图 3.4

我们在这个 **Sprite** 组件中单击第一行的 **Atlas** 按钮, 选择要创建的精灵所在的图集, 然后单击第二行的 **Sprite** 按钮, 会弹出这个图集所有的精灵预览界面, 从中选择所要的精灵。到此为止, 精灵就自制完成了。

关于创建精灵的方式我们就介绍这 3 种, 它的核心就是要为一个物体附加一个 **Sprite** 的组件。在制作过程中需要注意创建 **Sprite** 的节点, 也就是它应该在什么物体下创建, 这个涉及 UI 结构的设计, 我们在后面会详细讲解。

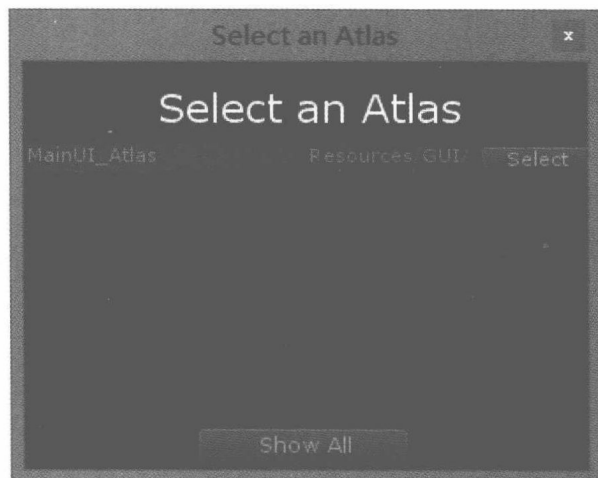
### 3.2.3 Sprite 组件的设置

我们参照图 3.4 所示的组件面板进行讲解。在 **Sprite** 组件面板中, 可以设置如下的一些参数。

(1) **Atlas**。单击 **Atlas** 按钮将会弹出图集选择界面, 如图 3.5 所示, 可以选择要使用

### 第3章 核心组件

哪一个图集（如果弹出的图集选择界面没有我们要的图集，记得单击该面板中的 ShowAll 按钮）。



▲图 3.5

（2）Sprite。单击 Sprite 按钮，将会弹出该图集所拥有的精灵的预览界面，我们只需要在其中找到需要的精灵，然后双击，就完成了设置。

（3）Type 和 Flip。在这里 Type 有 5 个选项：Simple（普通类型）、Sliced（切片类型）、Tiled（平铺类型）、Filled（填满类型）、Advanced（高级类型）。Flip 选项是翻转选项，相应的 Type 下有不同的设置。

- Simple

这种类型下，图片会正常显示出来，图片是什么样它就是什么样显示。当我们将一个精灵的尺寸拉大时，它会以原图拉伸（可能会导致原图发生形变）的方式来完成，如图 3.6 所示，我们将精灵的大小通过拉动四周的蓝色锚点拉大，精灵就被拉伸了。

在这种类型下，Flip 有几个选项，分别是：Nothing（不翻转）、Horizontally（水平翻转）、Vertically（竖直翻转）、Both（既水平又竖直翻转）。

这里的翻转和 Photoshop 中的图片翻转是一个意思，如图 3.7 所示。

- Sliced

切片风格，这种类型知识量比较大，和九宫格的制作联系比较紧密，所以在后文讲解九宫格的制作时，会详细讲解。





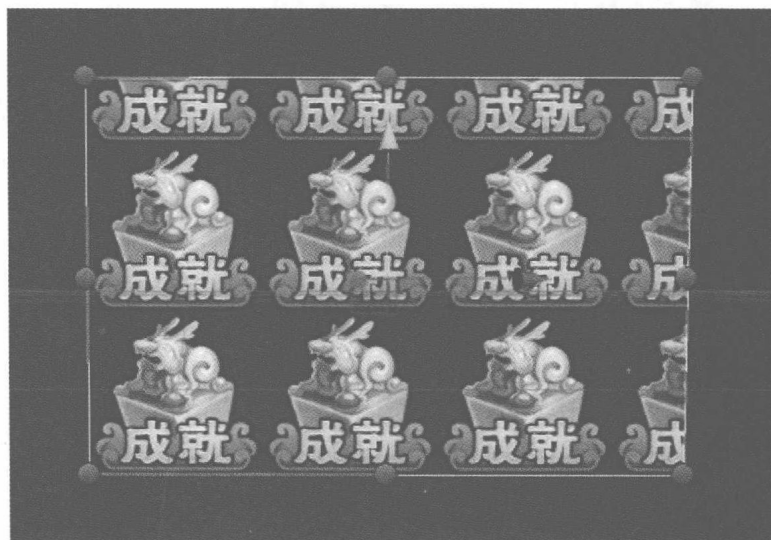
图 3.6



▲图 3.7

- Tiled

平铺类型，选择了之后，精灵尺寸会保持原来的尺寸不变，然后将精灵的尺寸拉大时，精灵会以平铺的方式来填充，并不会以拉伸的方式来填充。如图 3.8 所示，我们将精灵的大小通过拉动四周的蓝色锚点拉大，精灵变成了平铺模式。



▲图 3.8

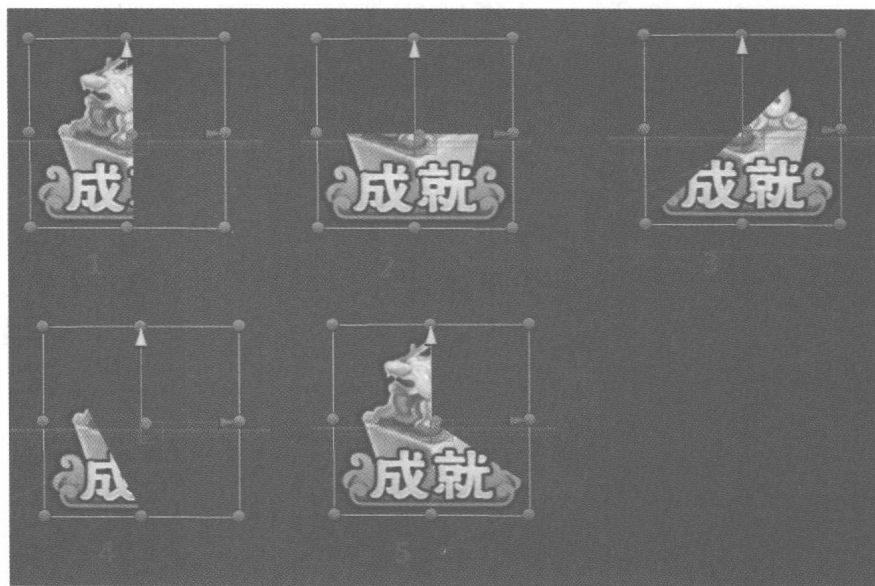
### 第3章 核心组件

#### • Filled

填满模式，这种模式可以设置图片填充一块区域的方式，例如，技能 CD 时技能图标前面有一层灰色的图片蒙住，这个灰色的图片要顺时针旋转消失，一直到转完为止灰色的蒙层彻底消失、图标恢复正常表示 CD 完成。

在 Filled 模式下，会多出 Fill Dir、Fill Amount、Invert Fill 3 个设置项。其中 FillDir 是指选择填充的方式，默认为 360° 填充。Fill Amount 可以设置填充的比例，默认为 1 全部填充。InvertFill 是设置填充的方向，不勾选是正方向，勾选是反方向。

如图 3.9 所示，图中 1 部分 FillDir 设置为 Horizontally 水平填充，FillAmount=0.5，InvertFill 不勾选默认为从左至右为正方向，图片相当于从左往右水平填充了 0.5，也就是 50%



▲图 3.9

图中 2 部分为 FillDir 设置为 Vertically 竖直填充、FillAmount=0.5，InvertFill 不勾选默认从下至上为正方向，图片相当于从下往上竖直填充了 0.5，也就是 50%。

图中 3 部分 FillDir 设置为 Radial90 填充（90° 旋转填充）、FillAmount=0.5，InvertFill 不勾选默认从右下角到左上角 90° 旋转为正方向，图片相当于从右下角到左上角旋转填充了 0.5，也就是 50%。

图中 4 部分 FillDir 设置为 Radial180 填充（180° 旋转填充）、FillAmount=0.25，InvertFill 不勾选默认从左下角到右下角 180° 旋转为正方向，图片相当于从左下角到右下角 180° 旋转

填充了 0.25, 也就是 25%。

图中 5 部分 FillDir 设置为 Radial360 填充 (360° 旋转填充)、FillAmount=0.65, InvertFill 不勾选默认为逆时针 360° 旋转为正方向, 图片相当于以中心点为中心逆时针旋转填充了 0.65, 也就是 65%。

- Advanced

高级设置因为比较复杂, 将会在后面讲九宫格的时候详细讲解。

#### (4) Widget 模块。

Widget 模块是 NGUI 的控件组件都具有的一个模块, 如图 3.10 所示。该模块的参数设置如下。



图 3.10

- Color

通过这里可以整体改变控件的颜色和透明度, 改变颜色的规则为: 原色值乘以这里设置的色值 (Unity 中, 会把 RGB 色值从 0~255 转变为 0~1 的一个浮点数)。

我们单击这个白色区域会弹出调色板, 如图 3.11 所示。可以随意地在这里设置控件的颜色值和透明度。

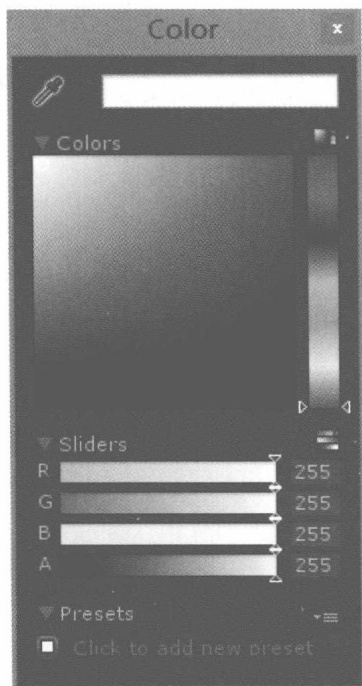
值得注意的是, 如果通过这个控件改变了透明度, 那么这个物体的子物体的控件透明度也会跟着被改变。

- Pivot

锚点设置, 默认为中心点。通过这一排按钮可以设置出左上、顶中、右上、中左、中心、中右、左下、底中、右下共 9 个点。

这个锚点设置, 改变的是图片的中心点位置, 这个 UI 控件和其他 UI 控件之间的相对位置就是以这个点作为标准的。





▲图 3.11

- Depth

深度设置，前文已经详细讲过。可以通过单击 **Back** 和 **Forward** 来减 1 和加 1，也可以直接输入一个深度数字来完成设置。

- Dimensions

尺寸，这里指的是控件的像素尺寸。单击 **Snap** 可以将图片的像素尺寸直接设置为原大小（这个图片被改成图集之前的图片大小）。

- AspectRatio

宽高比，**AspectRatio** 后面的数字为当前该控件的宽高尺寸比例。后面有一个模式选择按钮，默认为 **Free**，可为图片随意设置高和宽。这里除了 **Free** 以外，还有两个模式：以宽为基础、以高为基础。如果选择以宽为基础，那么图片的高度设置不论怎么设置都无效，都会以宽度和当前的宽高比计算得出。同理，如果选择了以高为基础，那么图片的宽度就无法被设置，它的宽度都会以高度和当前的宽高比计算得出。

### （5）Anchors 模块。

这个模块是控件位置适配的锚点设置，在后面讲解屏幕自适应时会详细说明。

## 3.3 制作标签 (Label)

### 3.3.1 怎样判断是否应当使用标签

当游戏中出现需要程序输出文字的地方, 就要使用标签。例如, 物品的名字千变万化, 无法提前预知有哪些名字, 就需要程序人员做一个 Label 来动态显示物品的名字。

### 3.3.2 创建标签

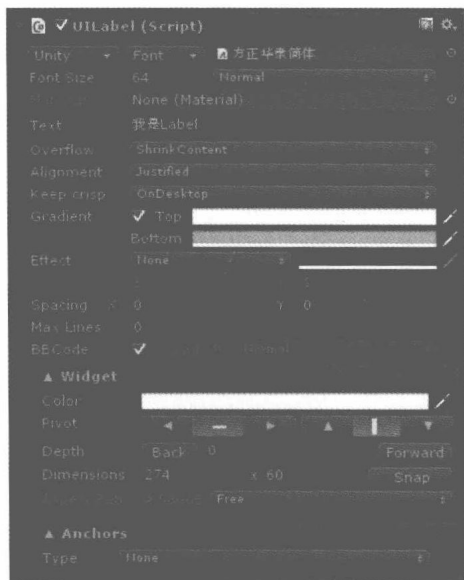
标签的创建方式也多种多样, 具体和 Sprite 的创建差不多, 这里就不浪费篇幅讲解了。一般来说, 可以直接在 Unity 顶部选则 NGUI 菜单、选择 Creat、选择 Label, 即可创建一个 Label。

### 3.3.3 Label 的文字设置

Label 组件的 Inspector 面板如图 3.12 所示, 我们来了解一下它的各项设置。

#### 1. 设置字体

如果新创建的 Label 的组件面板为一片灰色 (不可设置) 的话, 说明还没有设置字体。单击图 3.12 中 Unity 按钮, 会弹出两个选项: NGUI 和 Unity, 如果希望使用 NGUI 的静态字体, 则选择 NGUI; 如果希望使用动态字体, 则选择 Unity。



▲图 3.12

## 第3章 核心组件

然后单击 Font, 如果之前选择的 NGUI, 那么这里会弹出所有的静态字体以供选择。如果之前选择的是 Unity, 那么这里将会弹出所有的动态字体文件, 以供选择。

如果在其中没有找到你制作或者导入的字体, 记得单击 ShowAll。

### 2. 设置字号

可以在 FontSize 中设置我们希望文字的字号大小。但是文字真正显示出来的大小还要受 overflow 设置的影响 (下面讲解)。

在动态字体模式下 (选择的 Unity 中导入的字体文件), FontSize 后面有一个字体模式的设定, 默认为普通状态。其中可以设置字体为: Bold (加粗)、Italic (斜体)、BoldAndItalic (加粗并斜体)。

### 3. 设定字体内容 (Text)

我们可以在 Text 选项中, 输入需要它显示的文字, 支持回车换行。

### 4. Overflow 充满设置

要小心这个设置, 因为字体虽然设置了字号, 但是每一个 Label 其实依然是一个控件, 它也有尺寸。如果字体的字号大小导致字体超出了这个控件的尺寸, 这里的 Overflow 设置就会对字体进行处理。

- ShrinkContent

收缩内容。默认为这个选项, 意思为不管字体的字号设为多大, 只要它超出了这个控件的尺寸, 就将文字缩小到尺寸范围内。

- ClampContent

选择这个设置意味着如果文字的字号大小导致文字超出了控件的尺寸, 将不显示文字。

- ResizeFreely

选择这个设置意味着不管控件多大尺寸, 只要文字字号设定了, 文字会保持这个字号应有的大小, 然后控件会自动依照文字的大小调整宽高尺寸。

- ResizeHeight

选择这个和 ResizeFreely 类似, 只不过这个选项只会去自动调整控件尺寸的高度, 并不会让控件尺寸的宽度变大。



## 5. Alignment

这里是设置对齐方式，一共有：Auto（自动，一般会设为居中）、Left（左对齐）、Right（右对齐）、Center（居中）、justfied（调整，会自动变换）。

这里的对齐和居中的参照标准是控件的尺寸，也就是说左对齐，其实是对齐到这个 Label 控件的最左边。如果选择了 justfied，那么文字会在控件尺寸缩小到一定范围时，自动增大文字的间距来使文字刚好充满它。

## 6. Keepcrisp

字面翻译为保持脆性，默认为 OnDesktop。如果选择 Always，则当字体缩小时会变模糊，一般情况我们必须要去设置它。虽然能带来一些性能优化，但是非常渺小。

## 7. Gradient

梯度，可以理解为字体的渐变，默认为勾选状态。如果勾选，则字体从上到下会有一个渐变，在后面 Top 和 Bottom 两个色板中可以设置上部分和下部分渐变的颜色。如果不选择这个选项，那么字体将不再有渐变色，Top 和 Bottom 将不可用，此时字体的颜色将完全地以该控件的颜色为准。

## 8. Effect

字体的效果设置，一共有 3 个选择：None（无效果）、Shadow（阴影效果）、Outline（描边效果）。如果选择了阴影或者描边效果，可以在后面的色板中设置阴影或者描边的颜色，并可以在下面的 X 和 Y 中设置阴影和描边的 XY 厚度（约等于像素单位）。

## 9. Spacing

字体间距，可以设置 X（字间距）和 Y（行间距）的间距。

## 10. Maxlines

最大行数。后面的 BBcode 选项可以不用管。

## 11. Widget 和 Anchors

Widget 在 Sprite 精灵的讲解中已经讲过，这里是完全一样的。Anchors 在后文讲适配的时候一起讲。

## 3.4 制作 UI 纹理 ( UTexture )

### 3.4.1 什么情况下使用 UTexture

UTexture 的功能是在屏幕上显示一张图片，在这一点上它和 Sprite 有着相似的功能，但是 UTexture 会消耗单独的 DrawCall 去渲染，并会单独加载进内存，所以，会增大性能的开销。当我们判断是否应该使用 UTexture 时，可以遵循以下规律。

- (1) 当图片过大，不适合成图集时，可以使用 UTexture，此时要尽量保证图片的宽高是 2 的 N 次方（宽高不必相等，不过在 iOS 平台下必须宽高相等才能支持压缩）。
- (2) 当图片尺寸为 2 的 N 次方，但出现频率不高时，可以使用 UTexture。例如，游戏的 Logo，一般出现它都是在游戏开始的时候偶尔出现一下，此时可以使用 UTexture。
- (3) 修改更换特别频繁的图片，为了减少每次更新维护的麻烦，可以考虑使用 UTexture。
- (4) 如果图片很小，尽量将图片放入图集通过精灵的方式使用。

### 3.4.2 创建纹理

UTexture 的创建方式和 Sprite、Label 等一样，通过 Unity 顶部的 NGUI 菜单，选择 Creat 进行创建。其他创建方式就不做介绍了。

### 3.4.3 纹理的设置

UTexture 的 Inspector 组件设置界面如图 3.13 所示，我们来了解一下 UTexture 组件的各项设置。

#### 1. Texture

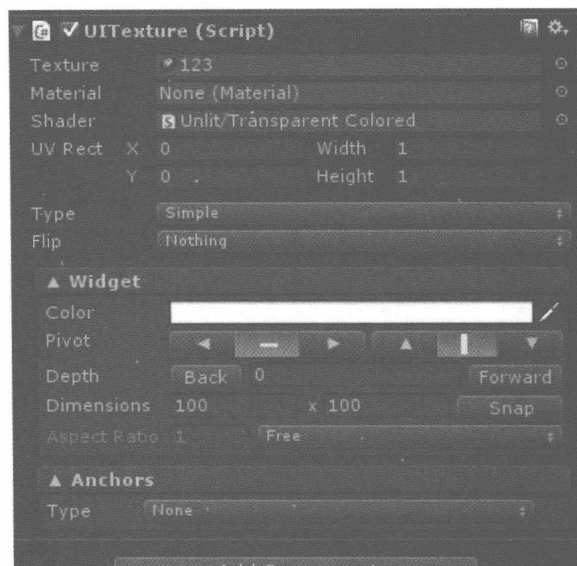
纹理设置，将要显示的贴图文件拖到此处即可完成设置。

#### 2. Material

材质设置，一般不用去设置它，如果有特殊材质需求可以拖到这里来。

#### 3. Shader

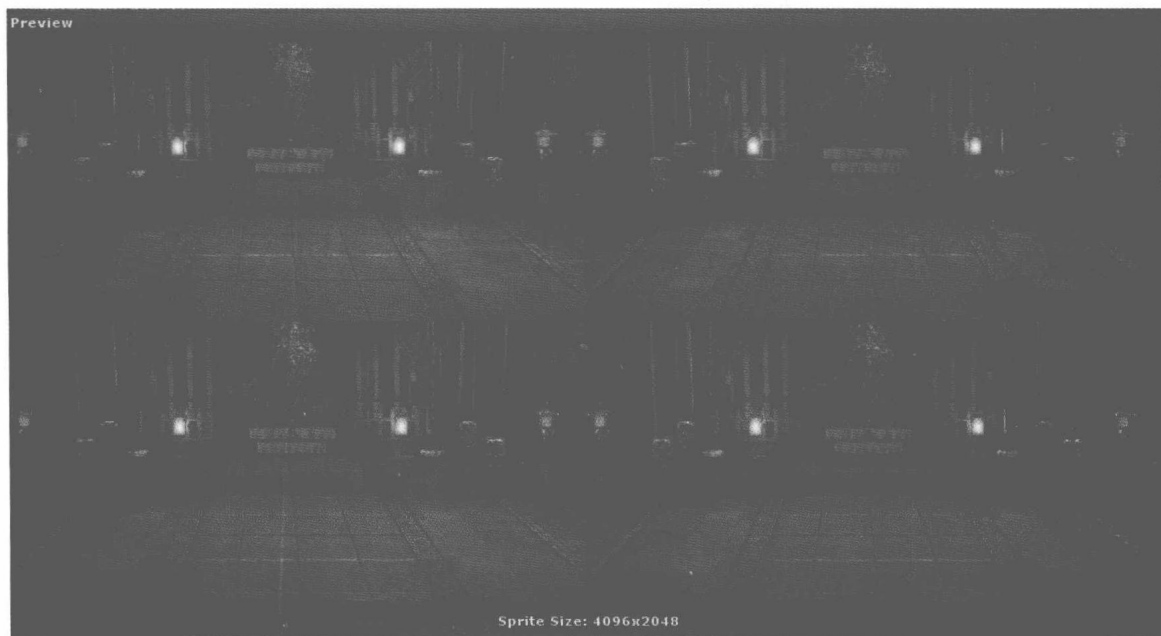
着色器设置，默认为带透明的颜色贴图着色方式，如果有特殊的着色需求，可以将其 Shader 拖到这里，不过，一些特殊的 Shader 将大幅增加性能开销，要谨慎使用 Shader。



▲图 3.13

#### 4. UVRect

UV 矩形的设置，如果在 width 和 height 中各填为 2，那么将会是 4 张纹理拼在一起。如图 3.14 所示，一般游戏开发中，这个 UVRect 都不需要进行设置。



▲图 3.14



## 第3章 核心组件

### 5. Type/Flip

这个和 Sprite 一样, 请参看前文对 Sprite 的讲解。

### 6. Widget/Anchors

略。

## 3.5 制作按钮 ( Button )

### 3.5.1 怎样判断应该使用按钮

按钮是我们制作 UI 过程中最核心、最重要的控件之一, 在一个游戏中, 按钮是用户操作最依赖的控件之一, 几乎无处不在。按钮的核心作用就是: 能接收我们的单击事件, 然后触发一个响应事件。

当我们在判断是否应该使用按钮时, 可以先了解按钮的核心作用。

(1) 按钮能接收单击并触发响应事件。

(2) 按钮被单击时能同时触发多个响应事件。

(3) 按钮可以有普通、悬停、单击、禁用等多个状态的不同表现。

(4) 不用去区别一个按钮是普通 Button 还是 ImageButton, 在新版 NGUI 中这俩已经合并, 只有一种 Button 了。

(5) 广泛的说, 按钮的核心在于接收事件, 任何可以接收用户操作事件的, 都可以称之为按钮。

图 3.15 展示了一些按钮的案例。



▲图 3.15

### 3.5.2 创建按钮

在旧版本的 NGUI 中,按钮分为了两种按钮:普通 Button (在一个背景底板上上面写有按钮的文字)和 ImageButton (按钮是一张纯图片,并且不同状态可以变为不同的图片)。旧版本中的按钮创建方式也是由 WidgetWizard 来创建。但是,将普通的按钮和 ImageButton 分开并不是很科学,因为在游戏开发过程中,这两种按钮的应用太广泛了,每次都需要去区分创建普通 Button 和 ImageButton 是一件很麻烦的事。所以,在新版本的 NGUI 中,普通 Button 和 ImageButton 合并了。

创建按钮,可以用以下任一种简单的方式。

(1) 创建一个 Sprite,这个 Sprite 将会是按钮的外形。

(2) 选中创建的这个 Sprite,然后在 Unity 顶部菜单中选择 NGUI、选择 Attach、选择 Collider。

(3) 选中创建的这个 Sprite,然后在 Unity 顶部菜单中选择 NGUI、选择 Attach、选择 ButtonScript。

我们可以用 Label、Texture 等其他控件来代替 Sprite 去制作一个按钮,方法一模一样,并不一定非要以 Sprite 作为基础进行创建。只是用 Sprite 制作按钮在游戏开发中更为普遍,所以以它作为例子。

如果需要在创建好的这个按钮上写上文字,例如,“确定”、“取消”等,只需要选中这个按钮,然后在它下面创建一个 Label,并写上“确定”即可,注意,Label 的深度要高于这个按钮的深度,这样就完成了一个“确定”的按钮。

小提示,创建出来的 Sprite 记得单击 Snap,让它回归到原尺寸大小,然后再去等比例调整它的大小,这样可以尽量减少图片的形变。

### 3.5.3 核心组件 BoxCollider

首先,按钮要接收单击事件,必须要有一个控件形态,它可以是 Sprite、Label、Texture 等。至于这个组件的使用,我们前文讲过了,就不浪费篇幅多讲了。我们直接讲解按钮的另外两个核心组件。

#### 1. 核心组件: BoxCollider

BoxCollider 是一个物理组件,准确地说是一个物理碰撞盒,所有的需要接收外部输入事件的(如单击、拖动等)UI,都需要拥有一个 BoxCollider,这个 BoxCollider 代表的是响应事件的范围。如果没有 BoxCollider,那么这个控件无论如何都无法接收到外部事件,这是 NGUI 的一

## 第3章 核心组件

个底层原则。

既然 BoxCollider 代表的是接收事件的响应范围,那么,如果我们将一个按钮的 BoxCollider 大小设为全屏幕,则单击屏幕上任何一个地方,都相当于单击了这个按钮。

我们可以通过上一节讲的 Attach 方法自动生成一个 BoxCollider,也可以通过 Inspector 面板中单击 AddCompent 手动附加一个 BoxCollider。区别在于:通过 NGUI 菜单 Attach 的 BoxCollider 的大小,会自动刚好包围控件的范围,而手动创建的 BoxCollider,大小需要手动去调整。

BoxCollider 的组件设置如图 3.16 所示,图 3.16 中按钮的边框变成了一个绿框,这个绿框就是 BoxCollider 的包围框,只要在 Inspector 面板中展开 BoxCollider 的组件,就能看到这个绿框,如果 Inspector 面板中 BoxCollider 组件的设置菜单被收起来了,则不会出现这个绿框。

在 BoxCollider 中,我们可以看到它的设置很简单,先来熟悉一下它的设置。

### (1) Is Trigger。

是否打开触发器,这个设置对于 NGUI 没什么用,它打开的作用是可以通过物理碰撞触发事件(如相撞爆炸等)。



▲图 3.16

### (2) Material。

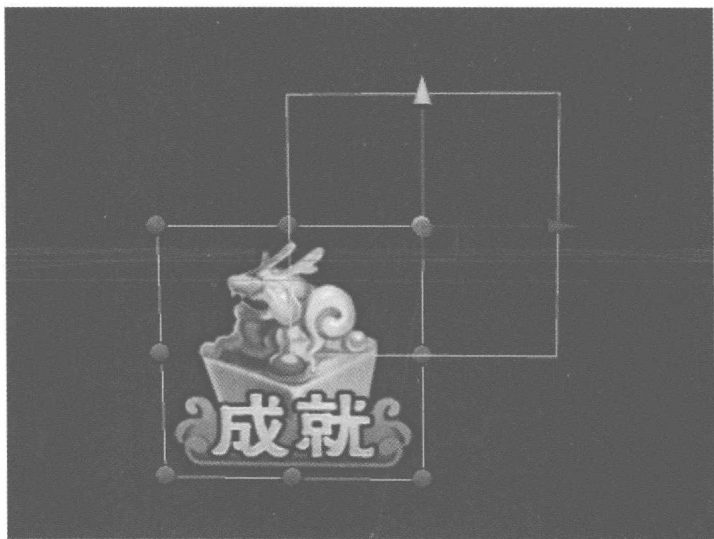
材质设定,这里设定的是物理材质,对 NGUI 也没有什么用,它的作用是为这个碰撞盒包



围的物体设定一个物理的表面, 例如, 一块地面是草地、还是木板、还是金属。

### (3) Center。

中心位置的偏移。BoxCollider 都有一个中心点, 这里的 Center 就是设置它的中心点相对于控件的中心点的偏移, 是一个相对量, 所以需要注意一点: BoxCollider 的这个 Center 会受到控件本身的 Pivot 中心点设置的影响。如图 3.17 所示的 BoxCollider 的 Center 偏移量为 (0, 0, 0), 但是, 因为控件的 Pivot 将中心点设置为了右上角的点, 所以形成了图 3.17 所示的情况。



▲图 3.17

### (4) Size。

尺寸设置。这是一个非常重要和常用的设置, 经常配合 Center 设置一起使用, 以此来调整控件响应区域的大小和位置。例如, 假设我们要做一款手机游戏, 在屏幕中有一个很小的关闭按钮, 经常导致玩家点不中它, 我们就可以依靠设置 Size 来将它的响应区域变大, 这样玩家只要单击到关闭按钮周围的区域, 都能触发关闭按钮的响应事件。值得注意的是, 这里 Size 的 Z 值在 UI 中是几乎没有用的, X 和 Y 的值都是以像素为单位。

需要注意的是, BoxCollider 一般需要依赖于一个非空的、实质的物体, 例如, 如果这个 BoxCollider 物体身上没有控件 (Sprite、Label、Texture 等), 只有一个孤零零的 BoxCollider, 那么在大部分情况下, 这个 BoxCollider 是无法接收事件的。

另一个核心组件如 3.5.4 节所讲。

## 第3章 核心组件

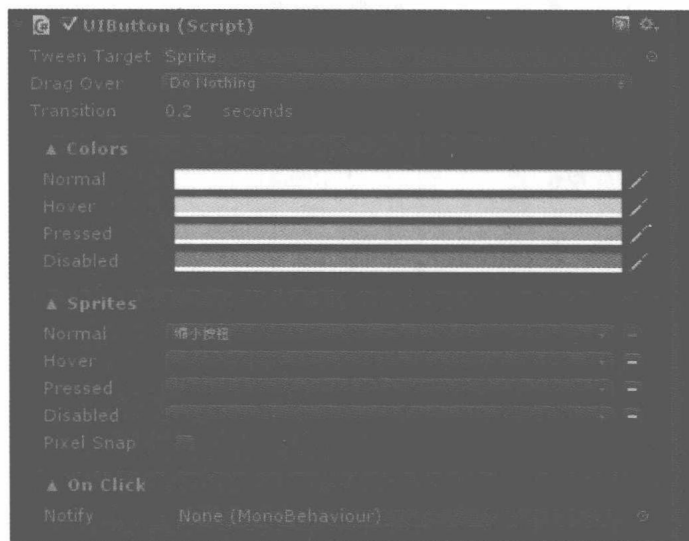
### 3.5.4 核心组件 UIButton

首先我们要明确的是，UIButton 并不是一个按钮的必须组件，也就是说，没有 UIButton 时，我们也能完成一个按钮。只要这个控件上有 BoxCollider，我们就可以在脚本中通过 OnClick()、OnHover() 等事件监听函数去触发一个响应事件。那么，UIButton 的价值到底是什么？什么情况下使用 UIButton？UIButton 有以下一些重要的用途。

(1) 可以设置不同状态下的颜色，比如普通状态、单击状态、鼠标光标悬停状态、禁用状态，可以有不同的颜色来表示。

(2) 可以设置不同状态下的图片，这就是所谓的 ImageButton，比如我们希望图片 A 在鼠标光标悬停在上面时，变成图片 B。

(3) 可以很方便地被动态赋予响应事件并分发事件。也就是说，我们可以临时地在 A 物体身上的脚本里动态让 B 按钮拥有一个或者很多个完全不同的响应事件。以前的旧版本 NGUI 中，都是用 EventListener 来实现，新版本中将会更加方便和快捷。



▲图 3.18

我们来了解一下按钮组件的设置，UIButton 按钮组件的 Inspector 面板设置界面如图 3.18 所示。

#### (1) TweenTarget。

动画目标，默认为按钮自己。按钮在光标悬停时变色、被单击时变换图片等，都是动画。绝大部分情况下，这个设置都需要设为自己（默认就是按钮自己）。

## (2) DragOver。

拖动结束事件，默认为 Do Nothing。这里有两个选项：Do Nothing 和 Press。之所以有这个选项，是因为按钮在被拖动时，有一个事件的交叉：如果我们拖动一个按钮，那么不仅仅拖动了它，同时也按下了它。这个设置的目的是，定义它被拖动结束后，是否还执行按下事件。

## (3) Transition。

过渡时间。这里是动画过渡的时间，例如，我们设定按钮在鼠标光标滑过时，要变黑，这个设置就是设置它在光标滑过时由正常到变黑的时间，

## (4) Colors 模块。

这里是改变颜色，可以设置在不同状态下按钮的颜色和透明度。一共提供了 4 种状态可设置：普通状态、按钮被鼠标光标滑过时的状态、按钮被按下的状态、按钮不可单击时 (BoxCollider 被禁用) 的状态。设置颜色的方式前文已讲过，就不再赘述。

## (5) Sprites 模块。

这里是精灵设置模块，也就是整合进来的 ImageButton 模块。值得注意的是，如果制作按钮时不是使用一个精灵控件作为基础制作的，那么将不会有这个模块。例如，我们为一个 Texture 附加 BoxCollider 和 UIButton 来制作按钮，此时 UIButton 组件就不会有这个模块。

在这里可以设置按钮在不同的状态下显示什么样的图片。一共支持 4 种状态：普通状态、按钮被鼠标光标滑过时的状态、按钮被按下的状态、按钮不可单击时 (BoxCollider 被禁用) 的状态。

当我们通过 Sprite 制作一个按钮时，我们创建的 Sprite 会默认出现在 Normal 设置中，然后任何状态下，按钮图片都是这个精灵不会再变化。

如果我们需要按钮在不同状态下进行变化，我们可以分别对每个状态需要显示的精灵进行设置，设置的方法为单击状态名称的按钮，会弹出图集中所有精灵的预览 (图集是 Normal 状态显示的精灵所属的图集)。

如果我们要取消一个状态下显示精灵的设置，可以单击该状态后面的小减号。但是，普通状态下的精灵无法被取消，当其他状态没有设置精灵时，将会默认显示普通状态的精灵。

PixelSnap 是指保持原像素尺寸，当我们设置了不同状态下显示不同的 Sprite 时，只有勾选了这个框，才会让显示的不同的 Sprite 都以原像素尺寸显示。如果不勾选，则所有状态下的 Sprite 都会统一以这个按钮控件 (制作按钮时所用的 Sprite) 的尺寸大小进行显示。



## 第3章 核心组件

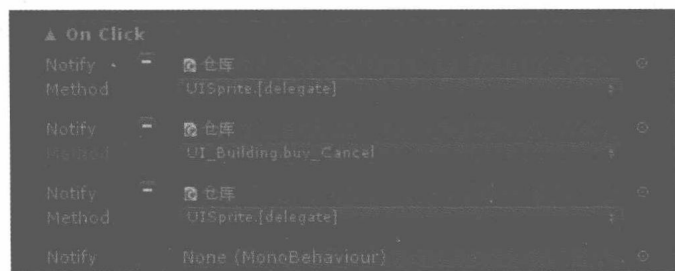
### (6) OnClick。

这里是设置按钮响应单击事件的地方。我们设置按钮响应事件时可以通过以下操作。

将该按钮要响应的事件函数的脚本所在的物体拖入到 Notify 中, 然后该物体的名称就会出现在 Notify 的设置框中, 会自动出现该物体下所有脚本中的 Public 函数, 然后选择要按钮响应的那个函数即可。

也就是说, 在这里可以让按钮响应任何一个物体身上的、任何一个脚本的、任何一个 Public 函数。

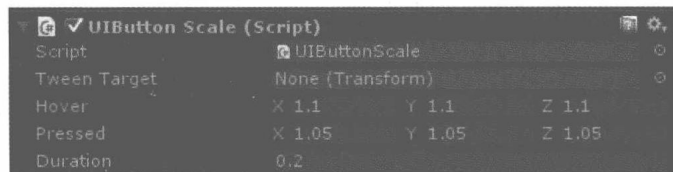
在这里当你设置了第一个响应事件之后, 会自动弹出第二个响应事件的设定, 也就是说, 通过这里的设置, 可以让按钮被单击之后响应任何多个事件, 如图 3.19 所示。



▲图 3.19

### 3.5.5 制作按钮的放缩动画

如果我们希望单击按钮时, 按钮会有一个放缩动画 (如突然变大并蹦一下), 我们可以单击 AddCompent、选择 NGUI、选择 Interaction, 在里面找到 ButtonScale 脚本, 附加到按钮物体上, 如图 3.20 所示。



▲图 3.20

ButtonScale 的核心作用就是控制按钮的放缩动画, 我们来了解一下它的设置。

#### (1) Script。

这个是它所调用的脚本, 我们不用管, 它会自动定位好。

## (2) TweenTarget。

这个是它所控制的动画作用的目标物体，我们不用管。在运行时，它会自动设定为当前所属的按钮物体。

## (3) Hover。

当鼠标光标划过时，按钮控件的大小变化。

## (4) Pressed。

当按钮按下时，按钮控件的大小变化。

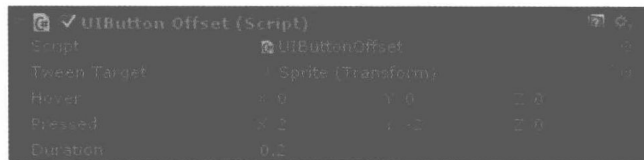
## (5) Duration。

完成缩放动画的时间，以秒为单位，默认为 0.2 秒完成。

需要注意的是，这个组件一般只用于按钮，如果非按钮要做动画，请参看后文讲解的 Tween 动画制作。

### 3.5.6 制作按钮的偏移动画

如果希望按钮有一个偏移动画，例如，单击按钮时按钮会向右下方蹦一下，我们可以为按钮制作一个偏移动画。方法和制作缩放动画类似，在 AddCompent 菜单中，选择 NGUI、选择 Interaction，然后找到 ButtonOffset，附加到按钮上。



▲图 3.21

图 3.21 就是 ButtonOffset 组件的设置界面，我们来了解一下它的设置。

(1) Script、TweenTarget 和 ButtonScale 组件一模一样，都不用管这俩设置。

## (2) Hover。

按钮在鼠标光标滑过时的位置偏移。这里偏移的是相对坐标。

## (3) Pressed。

按钮在按下时的位置偏移，这里也是相对坐标。

## 第3章 核心组件

### (4) Duration。

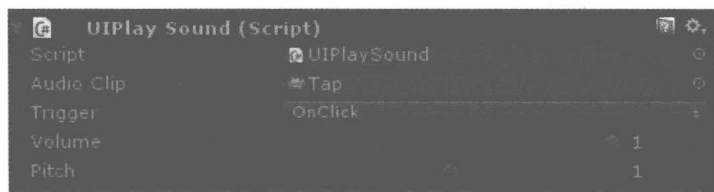
持续时间。

### 3.5.7 制作按钮的旋转动画

如果我希望按钮在被单击时能旋转一下,就可以为按钮制作一个旋转动画,方法和制作缩放偏移动画一模一样,旋转动画的脚本是 **ButtonRotation**。具体设置和其他动画类似,这里就不多赘述了。

### 3.5.8 添加按钮单击音效

在游戏开发中,一般情况下我们都需要为按钮增加单击音效,为按钮添加音效的办法很简单,可以在 **AddComponent** 中,选择 **NGUI**、选择 **Interaction**,然后将 **PlaySound** 组件附加到按钮上。



▲图 3.22

图 3.22 是 **PlaySound** 组件的设置界面,我们来了解一下它的设置。

#### (1) AudioClip。

音效的源文件,将音效文件拖到此处即可。

#### (2) Trigger。

触发模式,就是在什么情况下触发音效,默认为 **OnClick**。这里给了以下几种模式: **OnClick** (单击触发)、**OnMouseOver** (鼠标光标移上来)、**OnMouseOut** (鼠标光标移开触发)、**OnPress** (按下触发)、**OnRelease** (释放触发)、**Custom** (自定义触发,即脚本中控制触发)。

#### (3) Volume。

音量大小,为 0 到 1 之间的一个浮点数。

#### (4) Pitch。

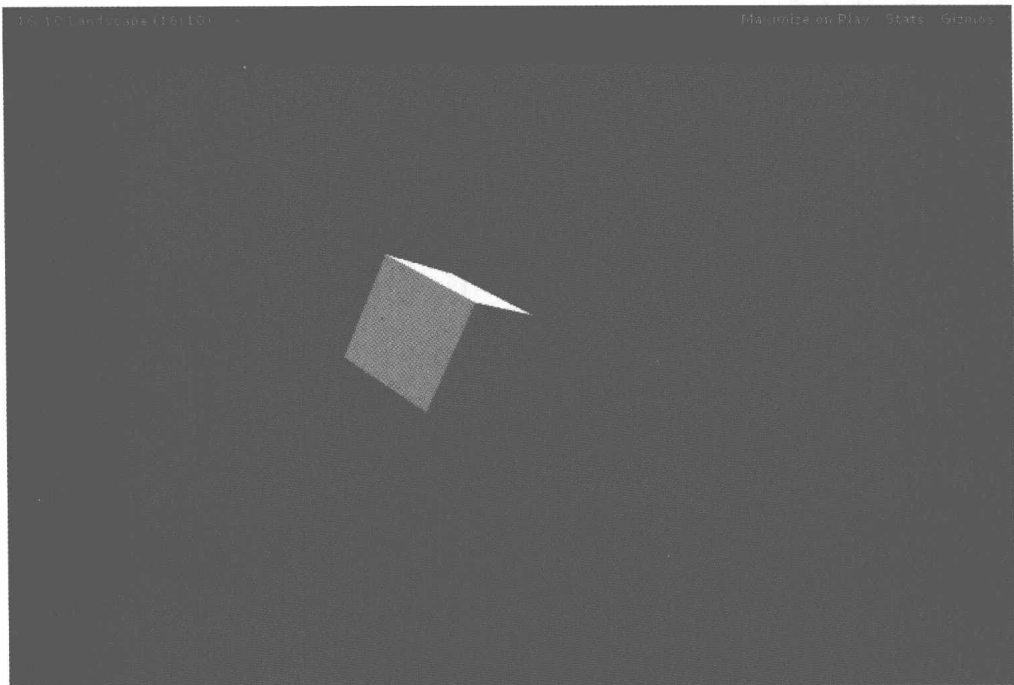
音调,也为 0 到 1 之间的一个浮点数。



### 3.5.9 任何事物都可以变成按钮，不仅仅是 UI

利用按钮的原理，可以将任何事物变成一个按钮。下面举个例子：我们把一个三维模型变成按钮，让它可以接收事件。

首先，我们单击 Unity 顶部菜单的 **GameObject**，选择 **Creat**、选择 **Cube** 创建一个立方体模型（也可以从外部导入模型文件，这里为了讲解方便，直接采用简单的立方体）。为了让它看得更清楚，可以加一个灯光，创建好之后如图 3.23 所示。



▲图 3.23

然后我们要进行如下的关键步骤。

#### 1. 让摄像机能够监听事件

根据前文所讲，只有带有 **UICamera** 组件的摄像机照射到的物体，才能接收事件响应。我们可以看到这个 **Cube** 是由场景中 **MainCamera** 渲染出来的，我们先为 **MainCamera** 附加一个 **UICamera** 组件。

选中 **MainCamera**，在 **Inspector** 面板中单击 **AddCompent** 按钮，选择 **NGUI**、选择 **UI**，然后选择那个 **NGUI Event System**，这个组件就是 **UICamera** 组件。这样我们就为主摄像机附加了 **NGUI** 的事件监听。

## 2. 为 Cube 附加按钮的关键组件 BoxCollider

选择 Cube, 首先为其附加一个 BoxCollider。可以通过 AddCompent 按钮选择 Physics, 然后 BoxCollider (也可以通过 Unity 顶部 NGUI 菜单去 Attach 一个 BoxCollider)。

默认情况下这个 BoxCollider 的尺寸为 0, 我们需要为它调整尺寸, 值得注意的是, 这个 Cube 因为不是 UIRoot 的子物体, 所以, 这里的尺寸单位是: 米。我们将尺寸设为 (1,1,1), 然后就能看到 Cube 被一个绿框所包围。设置尺寸的时候要注意, 这个 Cube 并不是一个 UI 图片, 所以一定要设置 Z 轴的尺寸。

## 3. 为 Cube 附加 UIButton

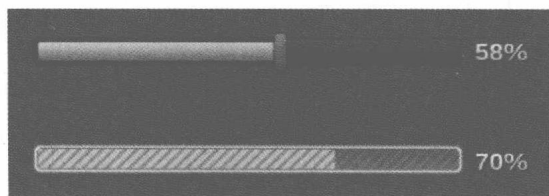
附加方法可以通过 Unity 顶部 NGUI 菜单去 Attach, 也可以通过 AddCompent。制作好之后, 我们会发现这个 Cube 之下并没有 ImageButton 才拥有的 Sprites 模块, 前文我们有讲过, 只有以 Sprite 作为基础制作出来的按钮才会有这个 Sprites 模块。

到此为止, 我们就已经将 Cube 变成了一个按钮, 鼠标光标移动上去, 会发现它颜色改变为 Hover 状态下的颜色了, 说明它已经接收事件了。

# 3.6 制作进度条 (UISlider)

### 3.6.1 怎样判断是否应当使用进度条

之前我们介绍了 Sprite、Label、Texture、Button 的用法, 之前的 4 种都是比较常用的基础控件。而进度条, 是一种更为高级的控件, 它是多个控件的结合体。用进度条的主要目的是为了用一根管子的充满程度来直观地表示某种数值的百分比, 进度条分为可拖动和不可拖动两种。图 3.24 所示为两种进度条, 上方的为可以拖动的进度条, 下方的是不能拖动的进度条。



▲图 3.24

可拖动进度条和不可拖动进度条的原理几乎是一模一样, 唯一的区别是可拖动进度条上多了一个拖动块和 BoxCollider 来接收事件, 而不可拖动的进度条只能显示一个数字的百分比, 无法由玩家去操控。

我们在判断是否应该使用进度条时, 有以下的规律可以遵循。

(1) 如果某一种值, 它有最大值, 我们需要表达它当前的值的占比, 这个时候用进度条会非常直观。此时应当用不可拖动的进度条。例如: 角色的生命值、法力值、角色升级经验等。

(2) 如果某一种值, 它有最大最小值, 我们希望玩家去自由拖动设置, 如音量调节、亮度调节等, 我们就可以使用可拖动的进度条。

需要强调的是, 可拖动进度条和不可拖动进度条, 他们的原理是一模一样的, 本质区别只是可拖动进度条有一个 BoxCollider。它们都有三大核心要素构成: 底槽 Sprite、进度条 Sprite、滑动块。

### 3.6.2 创建进度条

进度条因为是多种基础控件的一个集合体, 所以创建方式有多种。下面就介绍两种创建方式: 自己拼装和使用预设。

#### 1. 第一种方法: 自己拼装出一个进度条

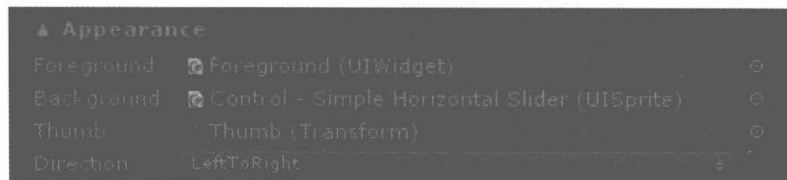
(1) 创建一个底槽 Sprite。

(2) 为底槽 Sprite 附加一个 UISlider 组件。附加方法为 AddCompnent → NGUI → Interaction → NGUI Slider。

(3) 在底槽 Sprite 下创建一个进度条 Sprite 作为子物体, 调整好尺寸用以和底槽相吻合。

(4) 在底槽 Sprite 下创建一个滑动块 Sprite 作为子物体 (如果不需要拖动可以不创建这个物体, 省去这一步), 然后在底槽 Sprite 上 Attach 一个 BoxCollider。

(5) 将底槽 Sprite 拖动到自身 UISlider 组件上的 Background 选项中, 将进度条 Sprite 拖动到底槽的 UISlider 组件上的 Foreground 选项中, 将滑动块 Sprite 拖动到底槽的 UISlider 组件上的 Thumb 选项中, 这样三大要素就齐备了, 如图 3.25 所示。



▲图 3.25

(6) 如果希望显示当前进度的百分比数字, 则在滑动块下创建一个 Label (如果不希望数



## 第3章 核心组件

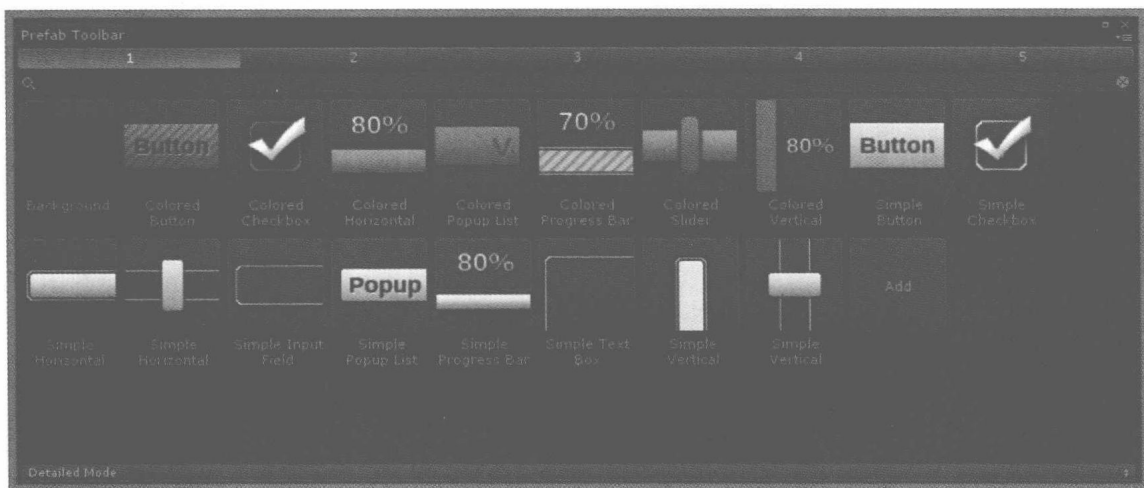
字的位置跟着滑动块走,也可以在别的地方创建 Label),然后将该 Label 物体拖动到底槽的 UISlider 组件的 OnValue Change 模块下的 Notify 中,然后在出现的 Method 选项中选择 UILabel.SetCurrent Percent 方法。

一个进度条到此就创建完成了。

### 2. 第二种方法:使用 PrefabToolBar 直接创建

在新版本的 NGUI 中,它自身制作了一些常用的 UI 控件的预设,当我们需要使用时,直接拖动预设到场景中,就可以直接完成创建。

在 Unity 顶部菜单中选择 NGUI 菜单,选择 Open、选择 PrefabToolBar,然后会弹出如图 3.26 所示的界面,这里面就是 NGUI 已经制作好的一些预设。



▲图 3.26

拖动其中想要的预设到 UIRoot 下(或者其他的 UI 节点下),就可以完成创建了。创建完成之后记得去修改它的各个 Sprite 为你所要用的 Sprite。

因为 NGUI 创建的预设是固定的,具有一些缺点,比如 UI 结构不一定符合我们实际项目的需求,比如某些组件我们并不需要。所以,当你足够熟悉理解和运用 NGUI 的情况下,在实际的游戏项目开发中,建议自己制作进度条。

### 3.6.3 核心组件 UISlider 设置

对于一个进度条控件来说,不管是不是可以拖动的进度条,它的核心组件是 UISlider 脚本,它的组件设置界面如图 3.27 所示:

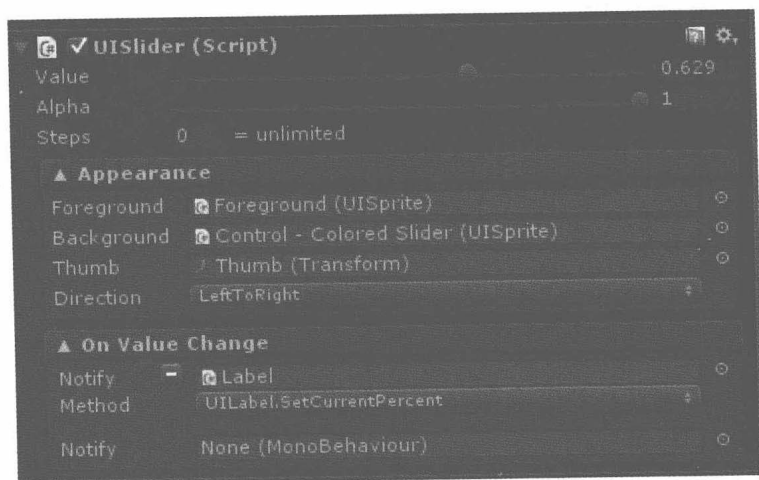
我们来了解一下它的工作原理。

### 1. Value

进度值，这是为了显示当前数值在“总槽”里的百分比，一般我们制作游戏时，都会在代码中调用这个参数，让它由游戏中相应的数据动态计算得出来。

### 2. Alpha

透明度，默认为 1。这里会控制整个进度条控件的透明度，可以用来做一些淡入淡出的效果。



▲图 3.27

### 3. Steps

每次变动的步伐大小。默认为 0，0 就是无限制，也就是 Value 值可以是任意一个值，如果设置了，那么 Value 就会“一段一段的”变化。一般游戏开发中，我们都不需要设置这个值。

它的填值效果为“关键点数量的概念”，例如，填入 5，则代表完整进度条只有 5 个点，相当于进度条的值将会只有：0、0.25、0.5、0.75、1 一共 5 个值。

### 4. Appearance 模块

这里是设置滑动条的一些组件。

#### • Foreground

这是进度条上层表示进度的 Sprite，将它拖动到这里就算完成了设置。Foreground 的长度会随着 Value 的变化而自动变化。

## 第3章 核心组件

### • Background

这是进度条的底槽 Sprite，将它拖动到这里就算完成了设置。底槽的长度是不会发生变化的。

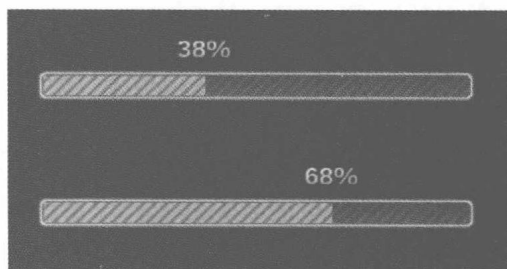
### • Thumb

这是拖动块的设置，将任何一个物体拖动到这里来，它就将随着 Value 的变化而发生位置的变化。

例如，如果希望在进度条的当前进度位置显示一个数字，就可以将这个数字作为 Thumb 拖到这里。然后数字的位置就会跟着进度条移动，永远处于进度条的当前进度位置。如图 3.28 所示，Thumb 数字的位置会随着 Value 变化，永远显示在当前进度的位置。

### • Direction

进度条的正方向，默认为从左至右。里面提供了 4 种选择：从左至右、从右至左、从上到下、从下到上。



▲图 3.28

## 5. On Value Change

这是进度变量发生变化时的一个回调函数，当 Value 值发生变化时，就会执行这里的函数。这里的设定方法，和设置 Button 的回调方法是一模一样的。

值得注意的是，如果我們希望在值发生变化时，自动改变一个百分比数字（Label）的显示，NGUI 为我们提供了一个简单的方法：将要显示该进度条百分比的 Label 物体拖入到 Notify 中，然后在 Method 栏中选择 UILabel.SetCurrentPercent 方法，这样，当进度条的 Value 值改变时，它就能自动地改变这个 Label 文本的显示。

### 3.6.4 进度条的 BoxCollider 说明

- BoxCollider 只有附加到底槽上才有用。



- 如果没有 BoxCollider, 进度条无论如何都无法进行拖动设置。
- BoxCollider 将会接收进度条上任何一个位置的消息来直接设置进度。例如, 不去拖滑动块, 直接在 90%的位置点一下, 那么进度会直接变为 90%。
- BoxCollider 和拖动块 Thumb 没有必然联系, 如果没有 BoxCollider, 那么即使有拖动块, 也无法通过拖动和单击等来设置进度。
- 只要有 BoxCollider, 即使没有拖动块, 我们也能直接拖动和单击来设置进度位置。

## 3.7 制作输入框 (Input)

### 3.7.1 怎样判断是否应当使用输入框

输入框, 就是用户可以自由输入文本的地方, 输入框因为其功能的单一性非常好判断是否应当使用输入框。当我们需要判断是否需要使用输入框时, 可以遵循一条原则: 凡是需要用户自主输入文本的地方, 几乎都必须使用输入框。

输入框的常见用法: 输入登录账号和密码、输入角色名称、输入聊天内容等。

### 3.7.2 创建输入框

我们来学习创建一个如图 3.29 所示的输入框。

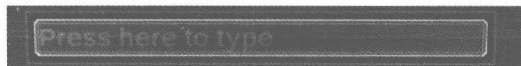


图 3.29

首先, 我们要了解输入框控件的 3 个核心控件: BoxCollider 组件允许 UI 能输入事件、UIInput 组件允许玩家能输入自己的文字、一个 UILabel 来显示输入的文字。

#### 1. 第一种创建方式: 自己拼装

(1) 创建一个 Sprite 作为输入框的底板 (就如图 3.29 中的底框)。

(2) 为这个输入框的底板附上 UIInput 组件, 附加方法为 AddComponent→NGUI→UI→Input Field。

(3) 为这个输入框附加一个 BoxCollider, 附加方法为先选中底板 Sprite, 然后选择 Unity 顶部 NGUI 菜单并选择 Attach→BoxCollider。或者 AddComponent→Physics→BoxCollider (此种方式创建的需要自主调整 Box 的尺寸用以匹配输入框大小)。

## 第3章 核心组件

(4) 在这个输入框下面创建一个 Label 子物体, 创建方法为选中输入框, 然后选择 Unity 顶部 NGUI 菜单, 选择 Creat→Label 即可。

(5) 将这个新创建的 Label 子物体拖入到底框的 Input 组件中的 Label 选项 (Input 组件的第一个选项) 中。

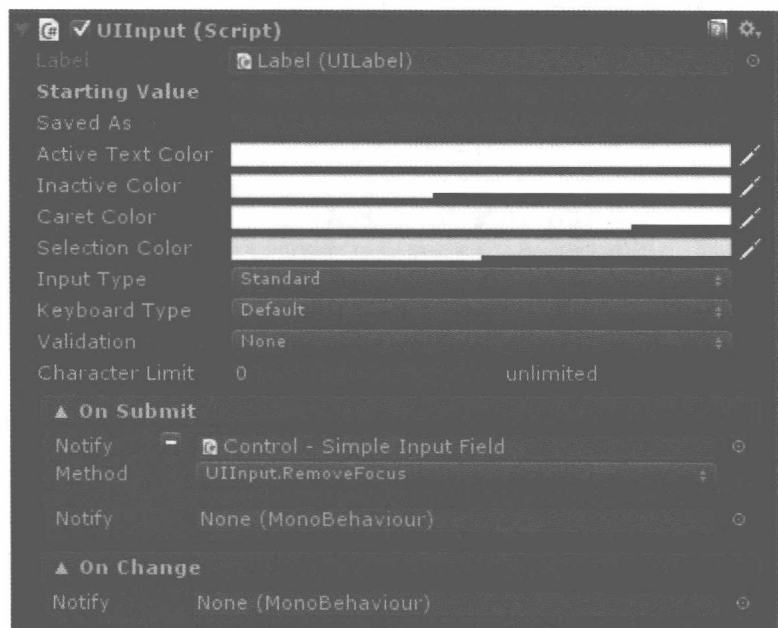
### 2. 第二种创建方式: Prefab Tool Bar 中直接拖入

之前我们讲过, 在新版本的 NGUI 中, 它制作了一批常用控件的预设。在 Unity 顶部选择 NGUI 菜单, 选择 Open→Prefab Tool Bar, 就能打开预设预览菜单, 然后在其中将名为 Simple Input Field 的预设体拖入到 Hierarchy 窗口中我们要创建的 UI 节点下即可。

### 3.7.3 核心组件 Input 设置

Input 组件是输入框的核心组件, 我们之前讲 Label 时说过 NGUI 显示文字几乎全部依赖 Label 组件, 所以, Input 组件它本身是无法显示文字的, 它必须和一个 Label 有一定的关联, 让这个 Label 来帮助显示 Input 的文本内容。

Input 组件的界面如图 3.30 所示。



▲图 3.30

我们首先来熟悉一下 Input 组件的设置内容。

## 1. Label 设置

这个是最核心的设置之一，刚才我们已经讲过，NGUI 中所有的文本显示都依赖于 Label，Input 本身是无法显示文本内容的，所以，它需要借助一个 Label 来显示玩家输入的文本，这也是我们之前在学习自己拼装输入框控件时为什么要在输入框下面创建一个 Label 子物体的用意。

我们将一个 Label 拖入到这里就算完成了设置，以后这个输入框中玩家输入的所有内容都将显示在这个 Label 上。

如果在这里我们不设置 Label，运行之后单击输入框将会报错，致输入框无法使用。

## 2. Starting Value

默认输入的文字。这里一定要注意区别“默认输入的文本”和“初始显示的文本”两个概念的区别！

默认输入的文本：当运行游戏时，输入框默认输入的文本，这个文本是模拟玩家输入进去的，就好像登录 QQ 时自动填充了你的 QQ 号一样，这个文本是一个属于输入的文本、真实有效的输入文本（只不过它是初始自动就输入进去了而已）。

初始显示的文本：这个是输入框在没有接收用户输入时显示的提示文本，例如：“请输入密码”、“请在这里单击进行聊天”等提示性文字，用户单击输入框之后文字就消失了变成了用户输入的文本的显示。这个初始显示的文本只有一个纯粹的提示作用，不属于输入的文本、是一个无效的文本。

这里的 Starting Value 变量设置的是默认输入的文本。初始显示的文本在 Input 相关联的 Label 中进行设置，这个关联的 Label 的文本内容就是输入框初始显示的内容。

## 3. Saved As

输入的内容在 Player Pref 中的哪个字段保存。这个我们一般用不到，而且它会自动保存，基本不用去管它。如果有特殊需求，可以在这里设置。

## 4. Active Text Color

活动文本的颜色和透明度设置，也就是用户在输入文字时的文字颜色和透明度。默认为白色，有需求的可以自行设定。

## 5. Inactive Color

不活动的文字颜色和透明度，这个可以用来设定初始显示的文字的颜色和透明度。

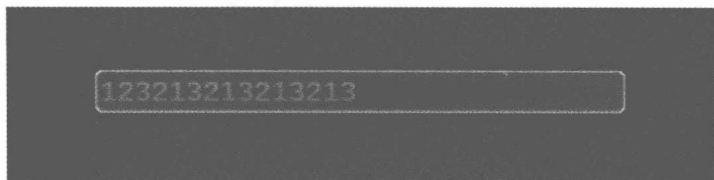


## 6. Caret Color

这个可以设定插入符的颜色和透明度。

## 7. Selection Color

选中的颜色和透明度，这个表示的是我们选中输入的文本时，覆盖在文本上的那一层遮罩的颜色，就像在 Word 办公软件中，我们输入的字可以通过拖动鼠标光标来选中它们一样。如图 3.31 所示，我们设定了选中之后遮罩为红色。



▲图 3.31

## 8. Input Type

输入类型的设定，默认为 Standard 标准输入。

### • Standard

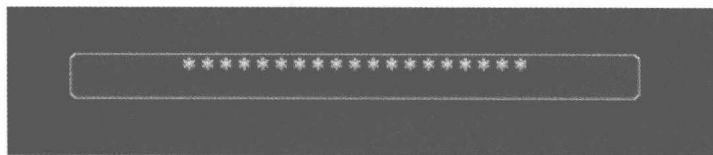
标准输入类型，输入的文字会正常一个挨一个地显示出来。它的对齐方式全部由 Input 相关联的 Label 决定。

### • AutoCorrect

自动调整模式，类似于 Label 中的自动调整模式。

### • Password

密码模式，在这种模式下，输入的文字会自动地变成\*符号，如图 3.32 所示。



▲图 3.32

## 9. Keyboard Type

这是输入文本时，键盘的类型设定。它有以下几种类型：

- Default;
- ASCCapable (任何格式都允许);
- NumbersAndPunctuation;
- URL;
- NumberPad;
- PhonePad;
- NamePhonePad;
- EmailAddress;
- HiddenInput。

## 10. Validation

验证。默认为 none 没有验证, 可以验证整数、浮点数等, 如果输入的字符不属于验证类型将无法输入 (整数允许视为浮点型)。

## 11. Character Limit

可输入的最大字符数限制。需要注意的是, 一个汉字要占用两个字符。

## 12. On Submit

这里是提交输入内容时的触发事件函数设定。

## 13. On Change

这里是当输入内容改变时的触发事件函数设定。

### 3.7.4 输入框使用的一些注意事项

输入框是一个集成了很多功能的组件, 我们在使用输入框控件时, 要注意以下一些要点, 以免发生不必要的困扰。

(1) 输入框 Input 本身是无法显示文字的, 它必须借助于一个 Label 来帮它显示输入的文本。

(2) 输入框输入文本的字体, 是在输入框关联的 Label 中设定的, 这个关联的 Label 用的什么字体, 则输入框中输入的内容就会是什么字体。

## 第3章 核心组件

(3) 输入框必须要有一个 **BoxCollider** 和一个 **Sprite** 底框, 否则无法输入。

(4) 输入框中对输入的文本的设定都受关联的 **Label** 的影响, 因为它本身是借助这个 **Label** 来显示文字的。但是, 如果发生冲突, 比如在 **Input** 组件中设定文字颜色为红色, 而在关联的 **Label** 中设定文字颜色为白色, 那么将会以 **Input** 中的设置为准。

(5) 如果发生以下情况, 都将造成输入框无法显示文字:

- 超出最大字符数限定了;
- 输入的字符不符合要求的验证类型;
- 关联的 **Label** 所选用的字体中没有这个文字;
- 关联的 **Label** 中设定了文字大小超出范围则不显示;
- 将文字设为全透明了。

(6) 输入的文字可以从 **Input** 中的 **value** 变量读取, 也可以从关联的 **Label** 中的 **text** 变量读取。

(7) 请将相关联的 **Label** 设为输入框的子物体, 这样就可以保证输入的文字和底框保持相对位置不变。

## 3.8 制作滚动视图 ( ScrollView )

### 3.8.1 怎样判断是否应当使用滚动视图

所谓的滚动视图, 是指一个可以滑动的视窗, 视窗大小和位置固定不变, 视窗内的内容用户可以通过手指滑动或者拖动滚动条来进行滚动浏览。比如说浏览器浏览网页时, 我们可以通过拖动右侧的滚动条来滚动浏览网页内容。

滚动视图的目的是为了解决同类内容过多, 一个 UI 版面显示不下的情况。如果同类内容过多, 我们一般可以采取设置多个页面, 然后通过翻页浏览的方式来浏览, 但是很明显, 滚动视图会比翻页更方便, 因为在移动设备上可以很方便地划屏进行滚动, 在 PC 上可以通过鼠标的滚轮进行滚动。

当我们需要判断是否应该使用滚动视图制作 UI 时, 可以遵循以下规律:

- (1) 有很多同类内容一个版面显示不完, 却必须要让用户很方便地进行浏览;
- (2) 它的核心目的是方便浏览。



### 3.8.2 创建滚动视图

滚动视图的创建方式有两种：通过菜单直接创建，或自己拼装。它们的基本原理都一样，都是为了凑齐几个核心组件。

我们来了解一下最简单的创建方式。

(1) 选择要创建滚动视图的 UI 节点，ScrollView 将会在这个 UI 节点物体下作为子物体进行创建。

(2) 单击 Unity 顶部 NGUI 菜单，选择 Creat、选择 ScrollView，这样将会自动创建一个 ScrollView 的子物体在选中的 UI 节点下。

(3) 到此，我们已经创建完成，我们可以将它的名字改为我们需要的名字方便管理。

目前为止只是创建的一个最基本的滚动视图视窗，现在还无法进行拖动看效果，因为滚动视图是一个相对复杂的 UI 控件组合，所以，将会分几个章节慢慢讲解它是如何工作起来的。

我们创建完的一个滚动视图的最基础的视窗如图 3.33 所示，图 3.33 中选中的那个红框就是滚动视图的视窗。



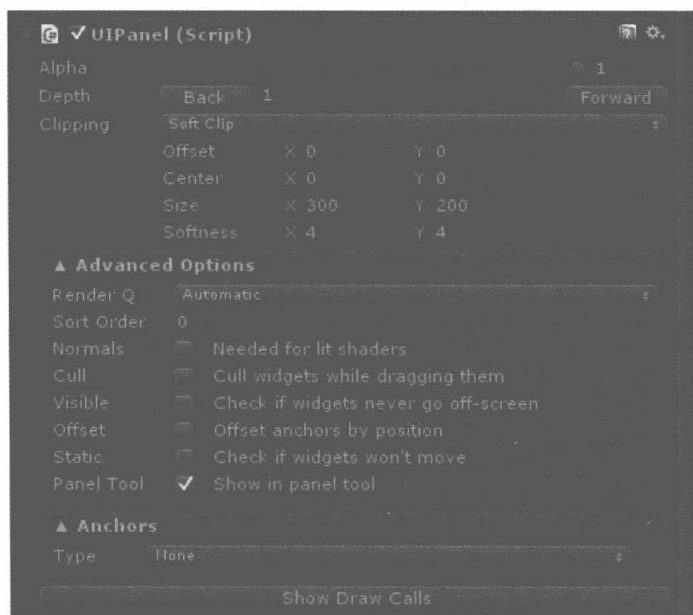
▲图 3.33

### 3.8.3 滚动视图核心组件 UIPanel

我们创建完滚动视图之后，发现它身上自动附带了 3 个组件：UIPanel、UIScrollView、Rigidbody。其中 Rigidbody 这是 NGUI 为了优化 UI 被移动等的性能开销而自动附加的，我们可以不用管它。而另外两个组件 UIPanel 和 UIScrollView 则是核心组件，它俩构成滚动视图的基本窗口。

## 第3章 核心组件

首先我们来了解一下 UIPanel 在滚动视图中的核心作用，它的组件截图如图 3.34 所示。



▲图 3.34

至于 UIPanel 这个组件的作用，我们在第 2 章讲解 NGUI 的几大基础核心组件时已经介绍过了，这里就不多赘述了。我们从图 3.34 中可以看到，我们创建的 ScrollView 身上的 UIPanel 组件有很多地方都自动设定好了，我们来一起了解一下它的工作原理。

(1) 在 ScrollView 被生成时，为了让它能够在上层显示，所以它自动给 Panel 设定了一个深度，这个深度大小是当前情况下刚好可以显示在最上层的深度，也就是当前 UIRoot 的 UI 树中深度最大的 Panel 的深度+1。

(2) 我们可以看到它的 Clipping 被自动设为了 SoftClip。之前我们讲过这是 UIPanel 的面板剪辑，也就是说 UIPanel 可以让自己的面板上只剪辑一块区域出来进行显示。这样在剪辑区域之外的 UI 元件就将看不见了。

Clipping 一共提供了 3 种模式。

### 1. None

无剪辑模式，这种模式下，滚动视窗中的物体可以被拖动，但是视窗因为没有剪辑，所以是没有边界的！这将可能导致内容被拖出屏幕外再也拖不回来。就像我们往下拖动浏览网页时会拖到一个所谓的“底”，none 模式就是没有这个“底”，你可以将内容全部拖出屏幕以外。

## 2. SoftClip

柔和剪辑模式，我们一般都会使用这种模式来制作 ScrollView。

在这种模式下，Panel 将会剪辑一块可视区域出来显示，这个被剪辑出来的区域以外的部分将会被剪辑掉而无法显示出来。

在柔和剪辑模式下，我们可以看到以下几个设置项。

- Offset

视窗的偏移，以像素为单位，设置这个参数将会导致视窗以 Panel 的中心点为基准进行偏移。

- Center

调整视窗的中心点，效果和 Offset 一样。

- Size

视窗的大小，一般情况下，我们都需要调整视窗的剪辑窗口的大小，以此来匹配背景底板的大小。如果视窗 Size 比底板大，将会导致视窗内容会滑动出底板的边框，如果比底板小，则会导致视窗内容滑动还没有到底板边缘就已经被剪辑掉消失了。

- Softness

剪辑边缘的柔和程度，视窗中，内容被拖动到边缘部分时，会有一个渐隐的效果。如果这个 Softness 的值设为 0，则内容被拖动到边缘时，会像被刀切掉一样被剪辑掉。

如图 3.35 中红框所示部分则为 Softness 的 X 为 50 的剪辑边缘，我们可以看到 X 方向的左右两边剪辑的边缘变得更柔和了。



▲图 3.35

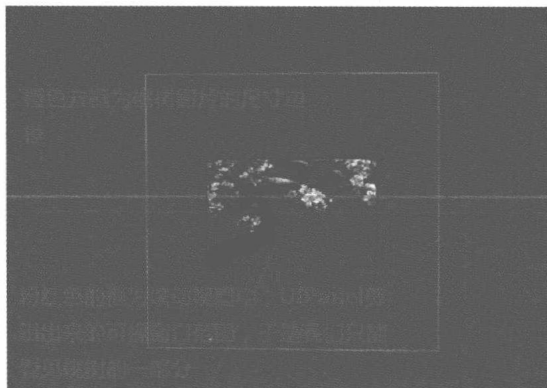


## 第3章 核心组件

### 3. Constrain but don't Clip

这种模式下是指视窗会尽量地包含所有内容但是不剪辑它们,效果大约等同于有边界但是边界为全屏,无法完全将内容拖到屏幕外面去,只要在屏幕范围内,都能看到内容,内容并不会被剪辑掉。

图 3.36 则是一颗巨大的桃树图片放在了一个 ScrollView 中,被 UIPanel 剪辑了的景象。通过这个图我们可以进一步加深了解 UIPanel 在滚动视图中的作用。



▲图 3.36

#### 3.8.4 滚动视图核心组件 UIScrollView

刚才我们了解了 UIPanel 是滚动视图中控制视窗大小的核心组件,而 UIScrollView 则是滚动视图中控制滚动功能的核心组件。没有 UIScrollView 组件的视窗是无法进行滚动的。

我们先来了解一下 UIScrollView 的组件,组件界面如图 3.37 所示。



▲图 3.37

## 1. ContentOrigin

这是滚动视图所包含的内容的起点，默认设置为左上角。

## 2. Movement

滚动视图的滚动方向，也就是内容的移动方向。一共提供了 4 种方式。

- Horizontal

水平方向拖动，也就是左右。

- Vertical

竖直方向拖动，也就是上下。

- Unrestricted

自由拖动

- Custom

自定义方向，会弹出新的 X 和 Y 设置框，可以通过对 X、Y 的设置来定义一个特殊的方向。

## 3. Drag Effect

拖动效果。里面提供了 3 种效果。

- None

无效果，拖到哪算哪。手指停下或者离开屏幕的一瞬间视图内容也停下了。

- Momentum

带动能的拖动，也就是有一个惯性，当我们手指松开时，视窗内容还会继续朝之前的方向滑动一会儿，中途如果遇到边界才会立即停下，否则会等惯性没了才会自动停下。这种效果主要目的是让用户用最少的操作来滑动视图内容。

- MomentumAndSpring

动能和弹性兼备的一种方式，这种方式和上一种动能方式很相似，区别在于，在这种方式下，当滑动内容拖到了视窗边界时，它可以被继续拖动“越界”，在松开手指时立即像弹簧一样弹回并回到正常视窗范围内。

## 第3章 核心组件

### 4. Scroll Wheel Factor

滚轮因素的设定, 这个值越大鼠标滚轮的滚动就会越灵敏。

### 5. Momentum Amount

动能因素的设定, 这个值越大, 滑动时的惯性就越大 (前提是 Drag Effect 有动能效果)。

### 6. Restrict Within Panel

选中后拖动将会被限制在 Panel 内, 这个是默认勾选的。如果不勾选这个选项, 则将会导致内容被拖到视窗的剪辑部分以外再也无法回来。

### 7. Cancel Drag If Fits

当它刚好适合视窗内时, 则自动退出拖动。

### 8. Smooth Drag Start

在开始的时候拖动平滑, 勾选上后, 我们拖动时内容会有一个速度从 0 变到我们拖动的那个速度的一种平滑自然的效果。如果不勾选, 在我们开始滑动时, 内容会瞬间与手指的速度一样, 就像黏在手指上了一样, 体验比较差劲。

### 9. IOS Drag Emulation

模拟 iOS 系统的拖动效果。这也是一种为了增强拖动体验的选项。

### 10. ScrollBars

为滚动视窗指定拖动条, 拖动条我们将会在下文介绍。这个设置是个非必选项, 可以设置也可以不设置, 不设置的话滚动视窗一样能正常工作, 因为在移动设备上, 我们可以用手指滑动, 在 PC 上我们可以通过按住鼠标键不放来拖动, 而拖动条, 可以理解为我们拖动的进度条, 就像我们浏览网页时最右边的那根竖直的进度条一样。

拖动条一共可以指定两个, 一个水平的、一个竖直的。

我们可以在 Show Condition 中设定拖动条出现的规则, 有以下 3 种规则可选。

- Always

不管什么情况, 滑动条总是出现。

- OnlyIfNeed



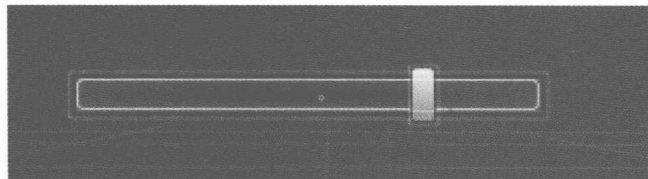
只有当需要的时候出现。那么什么时候是需要的时候呢？当我们的内容在滚动视窗内显示不完的时候，就会出现，如果我们在视窗内不需要拖动就可以看到全部内容，则不需要出现。

- WhenDragging

当拖动时出现。只要拖动内容，它就一定会出现。

### 3.8.5 创建一个拖动条

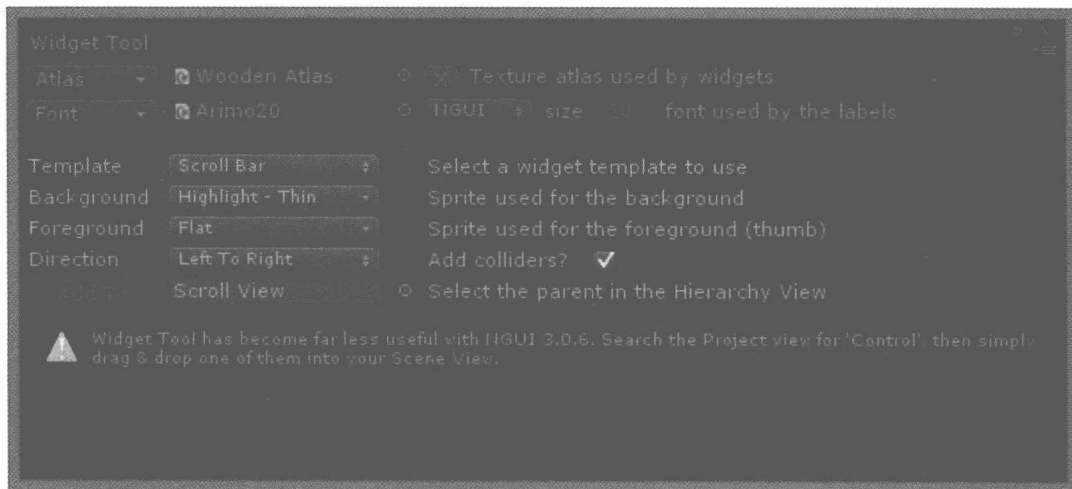
拖动条有多种创建方式，拖动条非常像一个进度条，鉴于拖动条的不同需求，它有非常多的创建方式，我们来简单介绍几种。拖动条一般只有水平方向的拖动条和竖直方向的拖动条两种形态，图 3.38 展示了一个拖动条的样子，这是一个水平方向的拖动条。



▲图 3.38

#### 1. 通过 WidgetWizard ( Legacy ) 创建

在 Unity 顶部 NGUI 菜单，选择 Open 选项，打开 WidgetWizard (Legacy)，会弹出 3.39 的界面。



▲图 3.39

这个界面我们在之前也讲过其用法，它是旧版本的 NGUI 的控件创建系统。在这个界面中，

## 第3章 核心组件

我们设定好图集和字体（如无需要可以不设定）后，在 **Template** 中选择 **ScrollBar**，然后在 **Background** 中设定它的底板的 **Sprite**，在 **Foreground** 中设定拖动块的 **Sprite**，在 **Direction** 中设定它的滑动方向。然后单击 **AddTo** 就可以在相应的 UI 节点下创建一个拖动块。

通过这种方式创建的滚动条我们需要调整它的 **Background** 和 **Foreground** 到我们需要的样子。

### 2. 通过 PrefabToolBar 创建

我们依然在 Unity 顶部 NGUI 菜单中选择 **Open**，打开 **Prefab Tool Bar**，弹出 NGUI 的预设界面，从中将“**Simple Horizontal**”（水平方向的拖动条）或者“**Simple Vertical**”（竖直方向的拖动条）拖动到场景中我们需要的 UI 节点下，即可完成创建。

通过这种方式创建的滚动条因为是预设体，所以需要调整它的大小和颜色。

### 3. 自己组装一个拖动条

拖动条在旧版本中是用 **ScrollBar** 组件来实现的，但是，在新版本的 NGUI 中，可以利用制作进度条的原理来方便地制作它。为了了解它的组件原理，我们还是需要学习一下怎样自我组装一个拖动条出来。

我们观察拖动条具有以下特点：

- （1）有一个底板槽来显示可以拖动的范围；
- （2）有一个滑动块，可以在范围内滑动。

通过对比我们可以发现拖动条和进度条的区别：拖动条就是一个缺少表层进度条的可拖动进度条！

所以，我们通过 3.6 节讲的方法，创建一个可以拖动的进度条即可，在制作这个进度条时，我们需要设置 **Thumb**，但是却掉 **Foreground** 的设置，一个拖动条就制作完成了。

#### 3.8.6 拖动条说明

通过不同的方式创建的滑动条有着不同的结构和不同的组件，可能会让我们迷惘，比如说我们通过 **Widget Wizard (Legacy)** 创建的滑动条和通过 **PrefabToolBar** 创建的滑动条功能都一样，但是结构完全不一样、组件也完全不一样。前者依赖于 **ScrollBar** 进行工作，后者依赖于制作进度条使用的核心组件 **UISlider** 进行工作。在这里我们通过如下说明来理清概念，避免混淆。

(1) **ScrollBar** 和 **UISlider** 我们观察组件界面可以发现, 它们非常相似。它俩创建的滑动条从本质上说工作原理是一模一样的, 没有任何分别, 所以不用去担心哪一种更正规哪一种更好, 因为它们都一样。

(2) **ScrollBar** 是旧版本的 NGUI 用来制作滑动条的, 在新版本的 NGUI 中, 随着 **UISlider** 的强大起来, 我们可以更方便地使用 **UISlider** 制作滑动条了, 所以推荐使用 **UISlider**, 也就是制作进度条的方式来制作滑动条。

(3) 我们依然推荐使用自己手动制作的方式去制作拖动条, 而不是使用旧的控件创建系统 (**WidgetWizard**) 和 **PrefabToolBar**。即使 **PrefabToolBar** 是新版 NGUI 的功能, 因为我们每个人的项目中的 UI 体系设计都不一样, 所以熟悉 NGUI 的控件原理后, 推荐自己去拼装控件。

(4) 切记, 滑动条就是一个没有 **Foreground** 的 **UISlider**。

### 3.8.7 让视图内的内容可以被拖动

我们从上文中已经学会了制作 **ScrollView** 和滑动条等知识, 但是到目前为止, 我们的滑动视图依然没有看到效果, 因为现在为止还不知道怎样去拖动内容。如果只是填充一些内容进去, 我们会发现, 即使有 **ScrollView** 也无法滑动里面的内容。这是因为需要对 **ScrollView** 所包含的内容进行一些配对设置, 以使它能够在 **ScrollView** 内被滚动起来。

如图 3.40 所示, 我们创建了一个 **ScrollView**, 在其下面创建了一个 **Texture** 子物体。子物体的 **Texture** 我们设置了一张巨大的桃树图片。从图 3.40 中可以看到, 桃树在滚动视图的范围内只能显示一小部分。

下面我们要让桃树可以被滑动以进行浏览, 这就是滚动视图的意义所在。

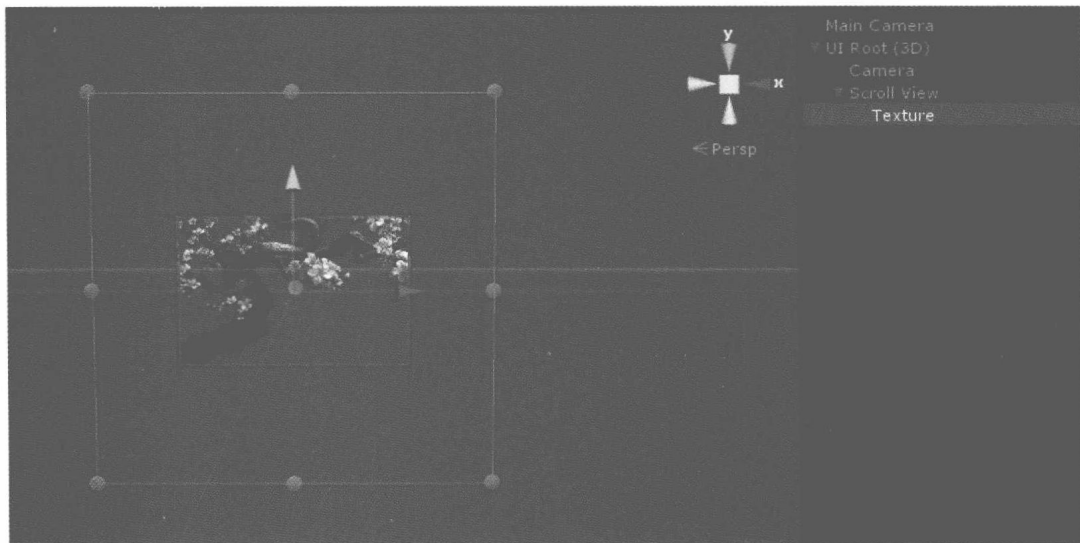
我们选中桃树的 **Texture** 物体, 首先为其 **Attach** (Unity 顶部 NGUI 菜单选择 **Attach**) 一个 **BoxCollider**, 因为, 既然我们需要拖动这个桃树的图片来查看, 就必须让桃树能接收到我们拖动的事件, 所以一定得有一个 **BoxCollider**。

然后为桃树的 **Texture** 物体附加一个核心组件: **Drag ScrollView**, 附加方式可以在 **Inspector** 面板中单击 **AddComponent**→**NGUI**→**Interaction**→**Drag ScrollView**。添加好组件之后无需进行任何设置, 运行游戏时它会自动地去读取父物体中的 **ScrollView**。

然后运行游戏, 拖动视窗内的桃树图片, 发现已经可以通过拖动桃树图片来进行浏览了。可以拖动的方向是 **ScrollView** 中 **Movement** 选项中设置的方向。



## 第3章 核心组件



▲图 3.40

### 3.8.8 制作滚动视图时的注意事项

滚动视图是一种比较复杂的控件类型，它需要多种控件配合使用，我们在使用时如果忘掉了一些制作的细节，则会导致 BUG 出现。所以，我们在制作滚动视图时，一定要深刻的理解它的工作原理，除此之外，还要牢记以下一些注意事项：

(1) 通常情况下，滚动视图一定要有一个 UIPanel 来进行窗口剪辑。这个 UIPanel 组件在创建 ScrollView 时会自动生成。

(2) 拖动条对于滚动视图来说可有可无，没有它滚动视图也能通过移动设备的触屏和 PC 设备的鼠标光标来进行滚动。

(3) 滚动视图内包含的内容，一定要有一个 BoxCollider，否则无法接收事件。

(4) 滚动视图内包含的内容，一定要有一个 DragScrollView 组件，这个组件会和 ScrollView 相互作用，在运行时，它会自动去找到父物体中的 ScrollView，然后和它相互作用，让视图内的内容可以被滚动起来。

(5) 滚动视图的内容，最好放到创建的 ScrollView 节点下面作为子物体存在，这样可以免去大量的烦恼和隐患。

(6) 滚动视图的滚动方向不要弄错了。

(7) 滚动视图的剪辑窗口的尺寸一定要调整到位，尽量别去调整 Clip 的 Center。



## 3.9 制作复选框 ( Toggle )

### 3.9.1 怎样判断是否应当使用复选框

复选框，就是对一个选项做上一个标记，表示这个选项已经被选中了。例如，我们在餐厅点菜时，经常会有服务员递给我们一个菜单，每一项菜品后面都有一个方框，需要点什么菜就在后面的方框内划上一个勾表示这个菜品选中了。在游戏中，复选框一般用来做一些选项的控制，这种选项一般都只有两种答案：是与否。例如，单击一下开启音乐的复选框，这个复选框上就打了一个勾，然后音乐在游戏中就会开启；如果再单击一下，则这个勾会取消掉，然后音乐将会在游戏中关闭。这就是复选框最常见的用法。

当我们要判断是否要使用复选框时，可以遵循以下规律。

- (1) 该功能只有两种选择状态：是、否。
- (2) 该功能同一时间只能激活且必须激活一种选择状态。
- (3) 该功能的两种状态为互斥关系，如果选择了一种状态，则自动关闭另一种状态。

简单地说，复选框其实就是一个开关，我们可以通过单击它来切换打开和关闭。一般开关的应用包括了选项勾选和页签。页签是复选框功能的一个高级应用，我们后文会讲。

根据以上的规律可以发现，游戏中很多地方都运用了复选框。例如，系统设置中，通过复选框功能来设定是否播放背景音乐。在角色界面中，通过复选框来确定是否显示时装，打钩则为显示时装，取消打钩则为不显示时装。

### 3.9.2 创建复选框

创建复选框的方法有很多，我们这里介绍两种方法：自己通过组件来拼装复选框控件和使用预设直接创建。

#### 1. 使用预设直接创建

NGUI 提供的预设中有复选框这个控件，可以从 Unity 顶部 NGUI 菜单中选择 **Open**，然后打开 **Prefabs ToolBar**，在界面中选中 **Colored CheckBox** 或者 **Simple CheckBox** 拖动到我们希望创建的 UI 节点下即可。

## 第3章 核心组件

### 2. 自己拼装复选框控件

(1) 首先选中我们希望创建复选框的 UI 节点, 通过 Unity 顶部 NGUI 菜单 Creat 中选择创建一个 Sprite, 然后设置它的图片, 让这个 Sprite 作为复选框的底框。

(2) 因为需要这个复选框来接收单击事件, 所以为这个底框 Sprite 制作一个 BoxCollider, 方法可以使用 Unity 顶部 NGUI 菜单中 Attach 一个 BoxCollider。

(3) 下面需要为这个复选框创建一个复选框的核心组件: UIToggle, 选中底框后, 通过选择 AddCompent→NGUI→Interaction→UIToggle 为这个复选框的底框附加一个 UIToggle 组件。

(4) 我们在这个底框的 Sprite 下面创建一个新的、表示选中状态的 Sprite, 比如一个勾。

(5) 我们将这个表示选中状态的 Sprite 拖动到底框 UIToggle 组件中 StateTransition 模块下的 Sprite 选项中, 然后将底框的 UIToggle 组件中的 StartingState 勾上。

到此为止, 复选框就算创建完成了, 运行一下可以发现默认情况下, 它是打钩选中状态, 单击一下则勾消失, 只剩一个底框; 再单击一下又打上了勾, 选中了。如图 3.41 所示, 左边表示选中状态, 右边表示未选中状态。

一般情况下, 我们都会在复选框的旁边写上一些文字, 表示这个复选框代表的什么选项。具体做法也是在底框物体下面创建一个 Label 作为子物体, 然后调整位置即可。



▲图 3.41

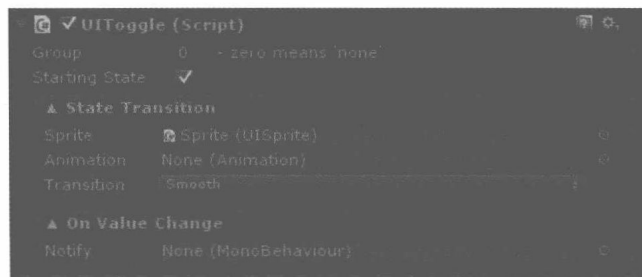
### 3.9.3 复选框的核心组件 UIToggle

在制作复选框时, 已经发现它的核心组件除了接收事件的 BoxCollider 以外, 就是一个我们新认识的 UIToggle 组件。这个组件的主要作用是一个开关, 通过它在打开和关闭之间进行切换。

UIToggle 组件的设置如图 3.42 所示, 我们先来了解一下它的设置。

#### 1. Group

开关组的设置。默认为 0, 表示没有开关组。



▲图 3.42

当有多个 Toggle 的 Group 相等且都不为 0 的时候, 表示它们在同一个开关组当中, 同一个开关组内的开关只允许打开一个。例如, 我们有开关 A、B、C, 它们的 Group 都为 1, 当打开 A 的时候, B 和 C 会自动关闭; 同理, 当打开 B 的时候, A 和 C 会关闭。

当一个 Toggle 不属于某一个开关组, 既 Group 为 0 时, 它就属于一个独立开关, 可以通过自身来进行开关, 当它属于某一个开关组时, 就无法通过单击自身来进行开启和关闭了, 因为一个开关组内必须要有一个开关是处于开启状态, 它必须通过开关组内部的多个 Toggle 之间的切换来进行关闭和开启, 例如, 刚才举的 A、B、C 3 个开关的例子, 如果 A 是打开的, 我们希望关闭 A, 则我们需要打开 B 或者 C 才能使 A 关闭, 因为 A 已经不是一个独立开关, 而是 ABC 开关组的一部分。

开关组的使用原理和一个非常常用的 UI 控件功能很像: 页签。页签也有着类似的功能需求, 有很多个页签, 我们同一时间内只能查看一个页签, 如果选择了一个页签, 则之前打开的页签自动关闭。

页签是 Toggle 的一个高级应用, 我们这里暂时只讲复选框的制作。

## 2. Starting State

是否初始状态, 如果选中, 则为初始状态, 不选中则不是。

这个设置项的意义在于: 当开关是一个独立开关 (Group 为 0) 时, 勾选 Starting State 意味着在初始状态下, 开关属于开启状态。当开关属于一个开关组时, 勾选 Starting State 则意味着在初始状态下, 这个开关组中初始处于打开状态的为这一个开关。

当一个开关组中有一个以上的开关都勾选了 Starting State 时, 则以这个组中排在最后的一个勾选了 Starting State 的 Toggle 为默认开启的开关。

## 3. State Transition 模块

这个模块是为了设置勾选的时候的一些关联 UI 表现。



## 第3章 核心组件

- **Sprite** 是设置选中状态下要显示出来的 **Sprite**，也就是之前创建复选框时创建的那个用来表示选中的打勾的 **Sprite**。设置的方法为，将表示选中的 **Sprite** 拖动到这个选项中即可。

- **Animation** 是设置状态切换时的动画，例如，需要选中时会有有一个勾从底框中蹦出来弹两下，那么就可以制作一个蹦出来弹两下的动画拖动到这个选项中。

- **Transition** 是选择开关切换时的一个平滑效果，里面提供了两种选项进行选择：**Smooth** 和 **Instant**，如果我们选择 **Smooth**，则在进行开关切换时，表示选中的那个 **Sprite** 的消失和出现会显得更加平滑一些；如果选择 **Instant**，则开关切换时，表示选中的那个 **Sprite** 的消失和出现会瞬间消失和瞬间出现。这个选项默认的设置是 **Smooth**，一般情况下 **Smooth** 的用户体验更舒服一些，所以，一般情况下我们不需要去调整它。

### 4. On Value Change

这里是设置当开关状态改变时触发的函数，设置方法和其他控件一样，就不再详细描述了。

## 3.10 制作下拉菜单 (PopupList)

### 3.10.1 怎样判断是否应当使用下拉菜单

下拉菜单，就是将一系列的选项隐藏，通过单击某一个控件将会弹出一个包含这些选项的列表，在其中选择想要的选项。这样做不但可以节省屏幕空间，也可以让用户在进行选择时更加方便快捷。

下拉菜单本质上还是一个单选框，与 **Toggle** 的功能有一些类似，对于下拉菜单玩家必须选择一个选项（有一个默认的初始选项），在同一时间也只能选择一个选项（单选性质）。在游戏开发过程中，如果碰到了以下特点的需求，就可以考虑使用下拉菜单了。

- （1）有一系列选项需要玩家做出选择，这些选项是有限多的。
- （2）这些选项玩家必须选择一个，也只能选择一个。
- （3）这些选项如果全部列出来用 **Toggle** 制作单选功能会非常占用屏幕空间。

如图 3.43 所示为一个典型的下拉菜单。

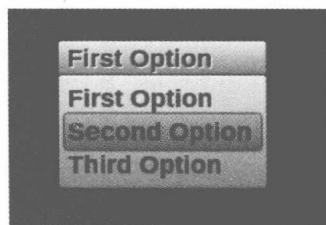
### 3.10.2 创建下拉菜单

我们可以有多种方式来创建下拉菜单，这里主要学习两种方式。



### 1. 第一种方法: 使用 NGUI 做好的下拉菜单预设体

在 Unity 顶部 NGUI 菜单, 依次选择 Open → Prefab ToolBar , 打开了 NGUI 内置的预设工具界面, 在其中选择 Simple PopupList 拖动到希望创建的 UI 节点下, 就算创建完成了。但是, 这种方法含有很多预定义的东西, 如果对 PopupList 的原理非常熟练, 那么不建议使用这种方法。



▲图 3.43

### 2. 第二种方法: 自我拼装一个

首先选中希望创建下拉菜单的 UI 节点, 在 Unity 顶部菜单, 依次选择 Creat → Sprite, 这样就在 UI 节点下创建了一个 Sprite 子物体。

然后在这个子物体身上添加一个 PopupList 组件, 添加方式为, 在 Inspector 面板中依次单击 AddComponent → NGUI → Interaction → PopupList。

因为下拉菜单需要单击, 所以, 还需要为它添加一个 BoxCollider 组件, 添加方法为选中这个控件, 在 Unity 顶部 NGUI 菜单, 依次选择 Attach → BoxCollider。

这样就算创建完成了, 组件截图如图 3.44 所示。



▲图 3.44

## 第3章 核心组件

可以在图 3.44 中红框所示部分输入一些选项，每输入完一个选项，就必须回车换行再进行输入下一行，因为 NGUI 会按行数来识别选项。我们在图 3.44 中输入了 1、2、3、4 4 个选项，运行游戏后，单击这个 PopupList 物体，可以看到弹出了一个下拉菜单，如图 3.45 所示。

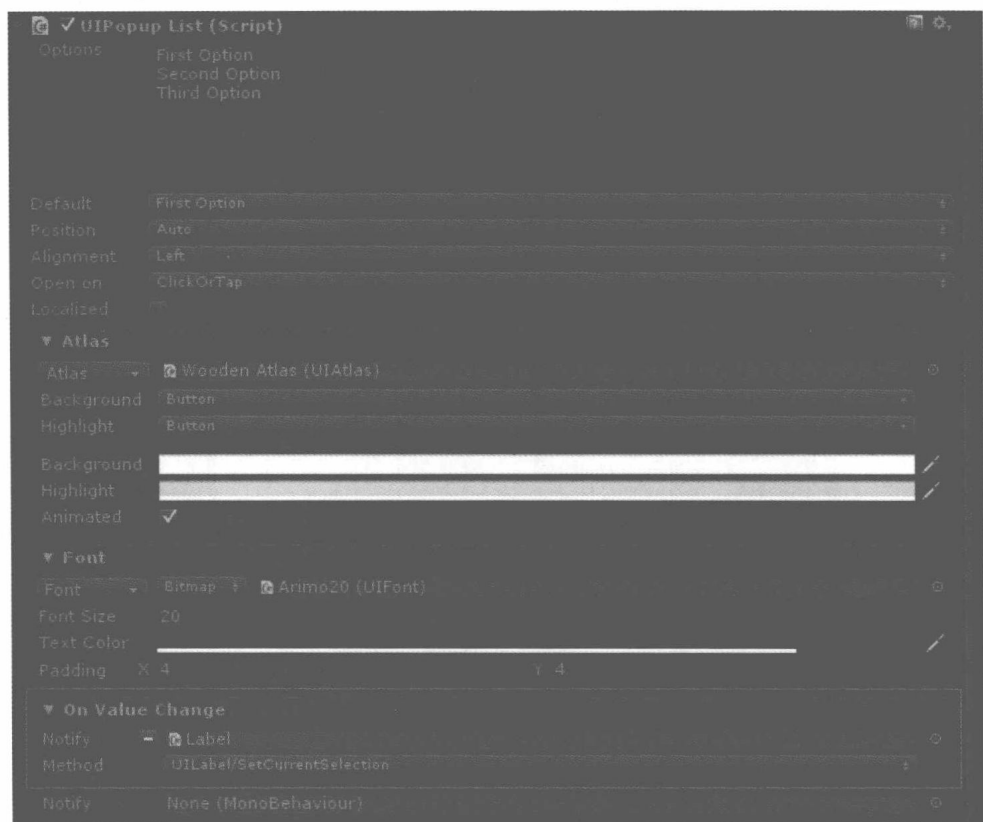


▲图 3.45

### 3.10.3 显示当前选中的选项

在图 3.44 中可以看到我们制作的下拉菜单并没有显示当前选中的选项，如果需要显示当前选中的选项，则可以在这个下拉菜单的控件下创建一个 Label 子物体，创建方法为选中这个下拉菜单的物体，在 Unity 顶部 NGUI 菜单，依次选择 Creat→Label。

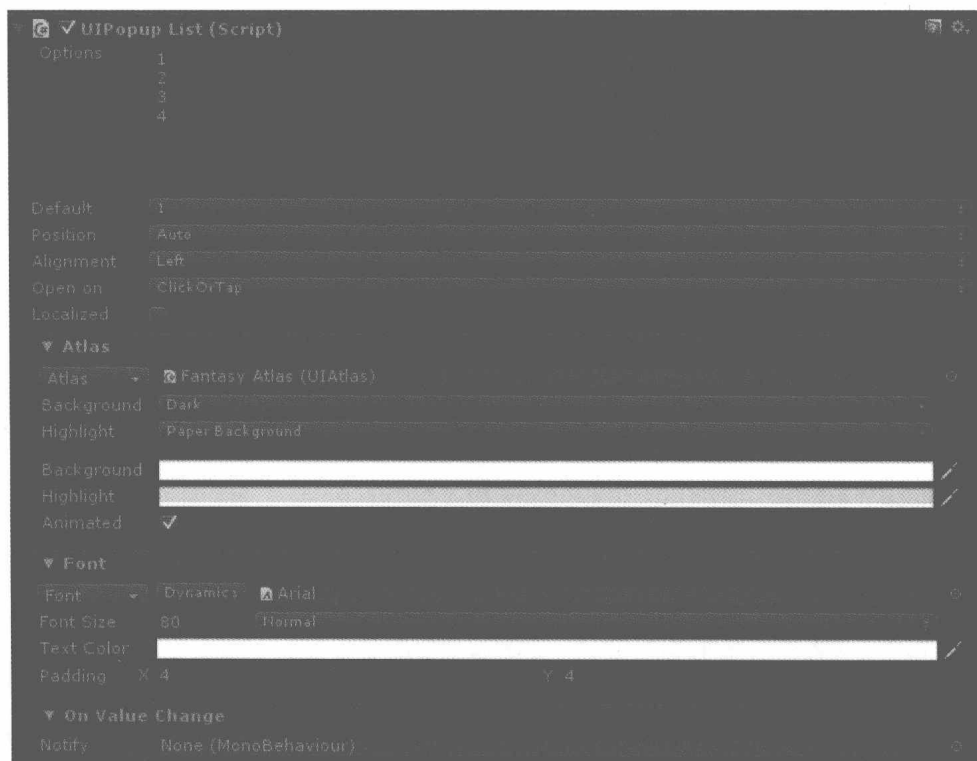
然后将这个 Label 和下拉菜单关联起来，我们将 Label 拖动到下拉菜单 PopupList 组件的 On Value Change 回调中，选择 SetCurrentSelection 方法，这样当 PopupList 的选项被改变时，当前被选中的选项会实时更新到这个被关联的 Label 上显示出来，如图 3.46 中红框部分所示。



▲图 3.46

### 3.10.4 下拉菜单核心组件 PopupList

制作下拉菜单需要一个 Sprite 来被单击, 需要一个 BoxCollider 来接收单击事件, 但是, 最核心的功能几乎全部依赖于一个核心组件: PopupList。这个组件的界面如图 3.47 所示。



▲图 3.47

- Options

这里是下拉菜单的各个选项录入的地方, 识别方式为按行识别, 也就是说每填入一个选项后, 需要回车换行才能继续录入下一个选项。

- Default

默认情况下选中的选项, 这个选项会自动填充为我们录入的第一个选项。

- Position

位置, 这里给了 3 个选项。

**Auto:** 菜单将会自动决定是从上方弹出还是下方弹出。

### 第3章 核心组件

**Above:** 菜单将会从上方弹出。

**Blow:** 菜单将会从下方弹出。

- **Alignment**

对齐方式，这里和 **Label** 里的对齐方式一样，就不多赘述了。

- **Open on**

打开的方式，这里提供了以下的方式可选择。

**ClickOrTap:** 单击出现菜单。

**RightClick:** 右键单击出现菜单。

**DoubleClick:** 双击出现菜单。

**Manual:** 手动出现，这种模式下任何输入都不会出现，必须代码控制它出现。

- **Localized**

这里是指菜单中的文本是否被本地化。

所谓本地化，可以理解为多语言翻译，例如，我们的游戏是英文的，那么游戏中的所有 UI 文本都将会被转换为英文版本。那么这个 **Localized** 选项就是决定这个下拉菜单是否被转换的。

如果打上勾，则表示这个菜单里的选项也会被本地化转换语言。如果不打钩，则表示这些选项里的文本不会被翻译，永远会保持它本来的样子。

为什么要有这么个选项呢？比如我们在选择语言的下拉菜单里，里面的每一种语言选项不能被翻译，因为玩家要依赖里面它认识的选项去选择自己要的语言。如果这里的选项被翻译成同一种语言了，那么就没有选择语言的意义了。

- **Atlas**

图集设定，这里有很多设置项。

**Atlas:** 选择图集。

**Background:** 设定下拉菜单的背景的精灵图片，还可以设置颜色。

**Highlight:** 设定下拉菜单出现后，鼠标光标移到选项上高亮显示时显示的图片，也可以设定颜色。



- Font

设定菜单文本的字体、字号大小等。

- On Value Change

当这个下拉菜单当前选中选项变化时，触发的事件。

可能我们会有疑问：为什么这个下拉菜单会单独进行是否本地化和使用 Font 字体的指定呢？这是因为 NGUI 考虑到大多数游戏在制作上有语言版本时，都会使用下拉菜单来让玩家选择它所认识的语言，这种情况下选项不能被翻译，使用的字体也可能很特殊要涵盖很多种语言文字，所以，NGUI 是一个非常贴心好用的插件。

### 3.10.5 制作下拉菜单的注意事项

我们在制作下拉菜单时，一定要注意以下事项：

- (1) 一定要有接收单击事件的 BoxCollider；
- (2) 填写选项时，一定要注意换行；
- (3) 如果制作下拉菜单是为了让玩家选择语言，则要更加注意本地化的设置和 Font 的设置。

## 第4章 UI 动画

### 4.1 常见的两种 UI 动画介绍

#### 4.1.1 要区分 UI 动画和 UI 特效两个概念

UI 动画的目的是为了让 UI 能够动起来，它的本质原理是通过每隔一定时间去改变 UI 的某个参数来实现动画效果。例如，每隔 0.5 秒就让 UI 的 Transform 组件的 Position 的 X 轴正向移动 1 个单位，持续 5 秒，那么就构成了一个 5 秒内 UI 朝 X 轴正向运动的动画。也就是说，UI 动画是在改变 UI 的组件参数，让 UI 实实在在地动起来了。

而 UI 特效，虽然效果上看着是 UI 特别绚丽的动态效果，但是，它的本质是叠加了一个特效在 UI 上，UI 本身是没有发生任何变化的。例如，我们经常在游戏中看到领取奖励的 UI 按钮在可以领取奖励时，会不停地闪烁发光，这其实是在领奖按钮上叠加了一层闪烁发光的特效，当可以领奖时，这个特效就激活。

所以，UI 动画和 UI 特效是两个概念，UI 特效具有独立性，可以通过叠加特效让 UI 看上去很绚丽，但是无法让 UI 本身动起来。而 UI 动画可以让 UI 本身动起来，却无法给 UI 赋予额外的特效效果。UI 特效需要额外的特效资源，UI 动画不需要任何的额外资源。

#### 4.1.2 关于 Tween 动画

Tween 动画是动画中一种非常常用，同时也非常简单的一种动画。Tween 动画简单一点的解释是：中间动画。它的原理是设定动画的起点和终点，以及这过程中每一个中间点的值，然后让物体按照设定的这个流程去插值地改变值。

例如，我们通过 Tween 动画来做一个物体的 X 轴位置改变的动画，设定该物体 X 轴位置

的 Tween 路线为 0、10、5 这 3 个点，设定好了之后，物体的 X 轴位置就会变化为 0，然后很平滑地变到 1、2、3……10，这个过程会根据时间计算出它变化的速度，然后一点一点地去改变，而不会瞬间变为 10，这就是所谓的插值。当物体的 X 轴位置变为 10 之后，它又会很平滑地慢慢变为 5，然后动画结束。

利用 Tween 动画可以做很多 UI 的动画效果，比如按钮被单击时会弹一下、按钮在某种情况下开始自己一蹦一蹦地上下浮动等。

在 NGUI 中，自带了一套非常精简实用的迷你 Tween 动画库，这个动画库虽然不能支持强大的 Tween 动画效果，但是，在 UI 的动画表现中已经非常实用。我们将会在后文详细讲解这个迷你 Tween 动画库。

### 4.1.3 关于 Animation 动画

Animation 动画就是 Unity 引擎自身的 Animation 动画系统，它的功能非常强大，从原理上来说，它和 Tween 动画几乎是一样的原理，通过插值去平滑地过渡每一个关键帧。但是，它支持 Unity 引擎中绝大部分组件的绝大部分参数，而且支持各种各样的运动曲线编辑，可以让动画按照一个自定义的、任意的节奏进行变换，类似于 Tween 动画的终极形态。

在 NGUI 中自带的迷你 Tween 动画库有一定的局限性，它只能实现一些常用的简单的动画效果，而 Animation 是 Unity 引擎原生的动画系统，非常强大。例如，我们希望让一个点光源的灯光不停地明暗变化，做出一种夜晚灯光闪烁的效果，或者天上星星闪烁的效果，我们使用 NGUI 自带的 Tween 动画库就难以做到，就需要求助于一些专业的 Tween 动画库，但是，我们也可以使用 Animation 轻松做到类似效果。

使用 Animation 来制作 UI 的动画主要是对付一些复杂的动画效果，一般情况下，优先使用 NGUI 自带的 Tween 动画，因为它简单，利于操作和维护。在碰到特殊情况下，才去使用 Unity 自身的 Animation 动画系统。

下面我们将逐一介绍 NGUI 的迷你 Tween 动画库中的一些常用的动画效果。

## 4.2 渐隐渐现动画（透明度动画）

### 4.2.1 透明度动画的介绍和应用

透明度动画的原理是改变 UI 控件的 Alpha 值来实现。在前文我们讲解 UI 控件的时候，讲到了很多 UI 控件都带有 Widget 的设置，如 Sprite、Label 等都有 Widget 的设置，在 Widget

## 第4章 UI 动画

设置里可以改变控件的颜色（其中包括了透明度）、尺寸、深度等。透明度动画改变的 Alpha 值就是改变的这里的颜色设置中的 Alpha 值。

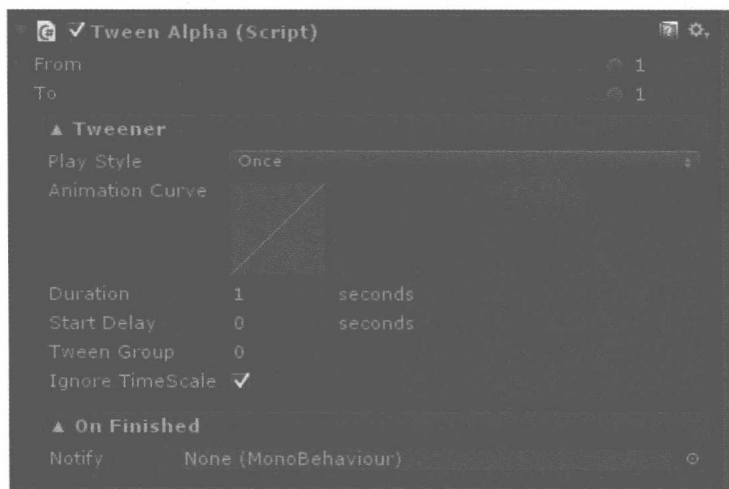
使用透明度动画可以做 UI 的渐隐和渐现，比如一个 UI 按钮出现时，它是慢慢地从透明变为不透明出现，消失时也是慢慢地透明掉。

我们也可以使用透明度动画做 UI 的若隐若现效果，例如，界面中有一行文字时而消失时而出现，不停地重复这个动画过程构成了一个若隐若现的效果。

### 4.2.2 使用透明度动画 TweenAlpha

我们需要为某个 UI 增加透明度动画时，可以通过为它增加一个 TweenAlpha 组件，这个组件在 NGUI 的 Tween 动画库中，添加方式为依次选择 AddComponent→NGUI→Tween→TweenAlpha。

添加成功后的 TweenAlpha 组件设置界面如图 4.1 所示，我们来了解一下如何设置它来制作一个透明度动画。



▲图 4.1

#### 1. From 和 To

这是 Tween 动画的核心设置项，起始点的设定。值得注意的是，在 NGUI 自带的迷你 Tween 动画库中，一个 Tween 动画一般只支持起始点的插值动画。

在 TweenAlpha 中，我们可以在 From 和 To 中设置动画的初始透明度和结束时的透明度，0 为全透明。1 为默认值，也就是不透明。



## 2. Play Style

这里是播放的风格，可以理解为循环模式，一共有以下 3 种形式。

- Once

只播放一次，播放完了之后，就停止。

- Loop

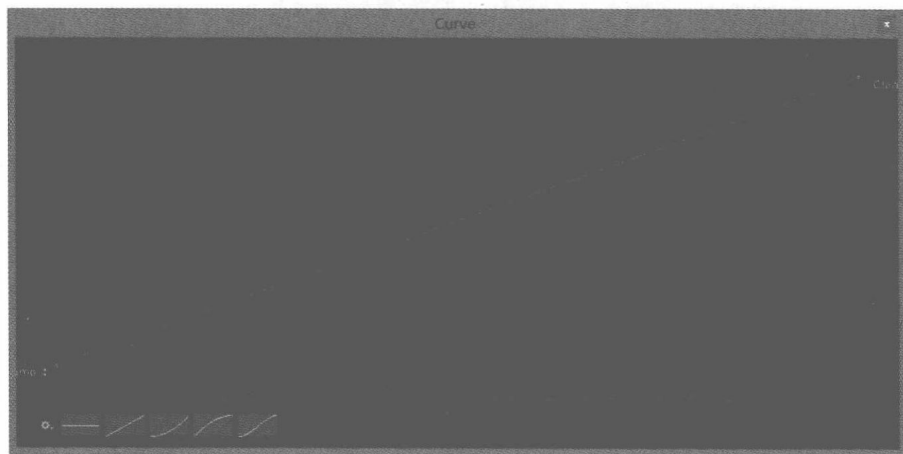
循环播放，播放完了之后它会立即瞬间回到起点接着播放。我们举一个比较形象的例子：有一个动画是从透明度为 1 变为透明度为 0，那么 Loop 模式下，动画播放时透明度的变化为  $1 \rightarrow (\text{中间插值}) \rightarrow 0 \rightarrow \text{瞬间变为 } 1 \rightarrow (\text{中间插值}) \rightarrow 0 \rightarrow \dots$  如此无限循环下去。注意：这里从 0 慢慢插值达到终点 1 之后，是瞬间变回 0 进行下一遍播放的。

- PingPong

乒乓模式，顾名思义，动画会像乒乓球一样来回播放，它和 Loop 最大的区别在于，Loop 在播放完了之后是瞬间回到起点然后继续播放，而 PingPong 模式在动画播放完了之后，会倒着插值播放。还是那个例子：有一个动画是从透明度为 1 变为透明度为 0，在 PingPong 模式下，动画播放时透明度的变化为： $1 \rightarrow (\text{中间插值}) \rightarrow 0 \rightarrow (\text{中间插值}) \rightarrow 1 \rightarrow (\text{中间插值}) \rightarrow 0 \rightarrow \dots$  如此无限循环下去。

## 3. Animation Curve

这是动画的曲线编辑，可以编辑一些动画的节奏。我们单击这个设置项中的曲线图片，会弹出以下界面，如图 4.2 所示。



▲图 4.2

## 第4章 UI 动画

从图 4.2 中可以看到一条默认的曲线，为一条直线。它是单次动画内的曲线编辑，直线的意思是起点的值会在持续时间内，非常平均地进行插值改变，最后变为终点。我们在图 4.2 中底部的几个小灰块中可以看到，有几条预设好的曲线供我们选择，熟悉函数的应该都明白它的意思，我们只需要单击其中一种曲线，即可将这条预设好的曲线作为我们需要应用的曲线。

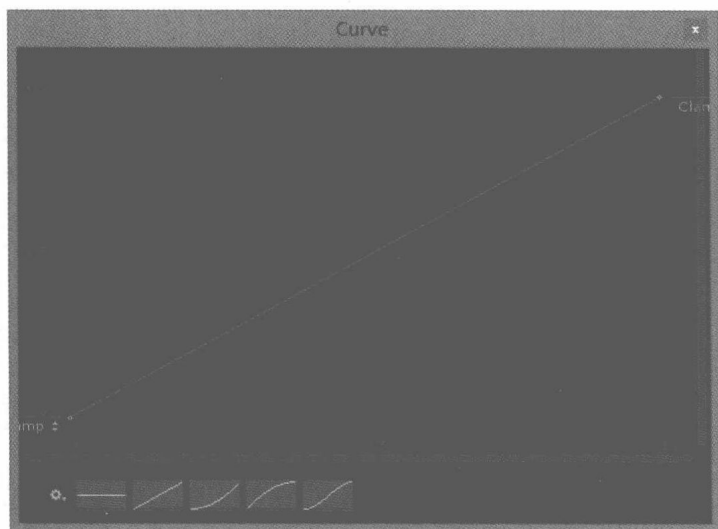
我们简单介绍一下几种预设好的曲线。

如图 4.3 所示，为水平的一条横线，这种模式比较特殊，代表着没有插值动画，动画在过渡时，不论动画需要过渡多少时间，它都会一瞬间跳到终点，完全没有中间过程，动画一旦开始播放就会立即达到终点播放完成。



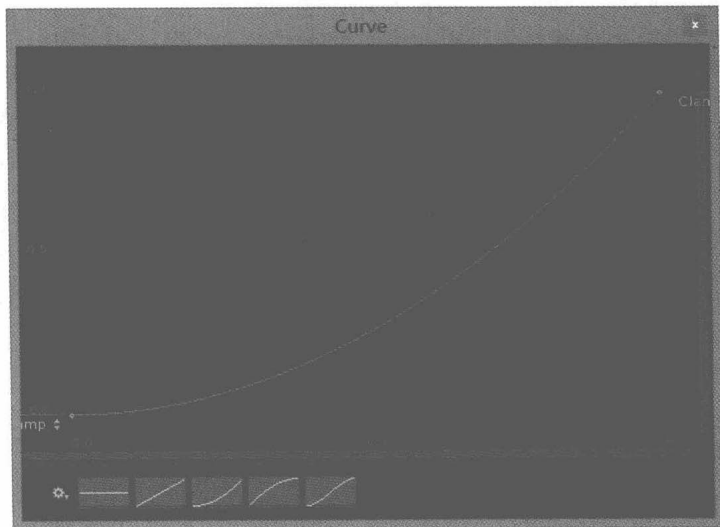
▲图 4.3

如图 4.4 所示，为直线，动画将会匀速从起点过渡到终点。



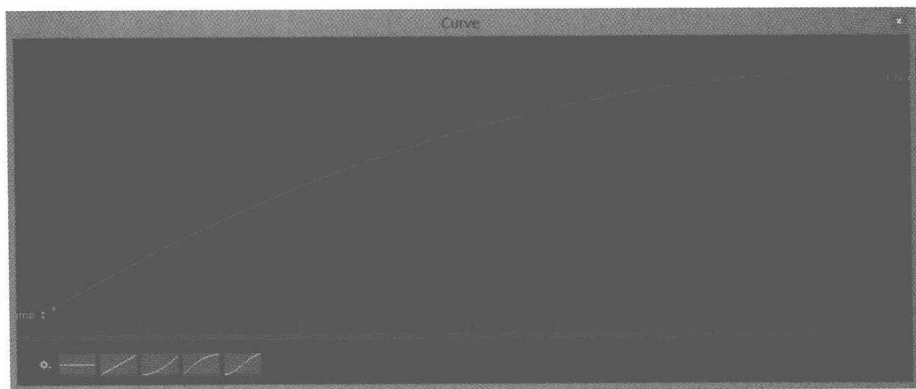
▲图 4.4

如图 4.5 所示，为先慢后快的曲线，动画将会以先慢后快的方式从起点过渡到终点。



▲图 4.5

如图 4.6 所示，为先快后慢的曲线，动画将会以先快后慢的方式从起点过渡到终点。

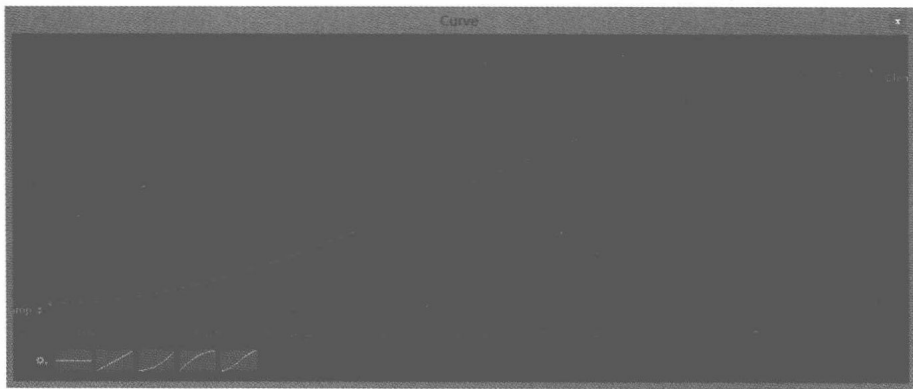


▲图 4.6

如图 4.7 所示，为先慢后快最后再慢的方式，动画过渡时，在开始时比较慢，然后变快，快到达终点时又开始变慢。

如果我们对它预设的曲线还不满意，我们可以在曲线中进行拖动设置，方法和 Unity 的 Animation 曲线一样（这个属于 Unity 的原生系统我们就不多讲述了）。我们也可以通过单击预设曲线左边的齿轮符号，在弹出的预设曲线界面中单击 New 来新创建一个预设曲线，以便于以后我们更方便地应用自定义的曲线。

## 第4章 UI动画



▲图 4.7

### 4. Duration

持续时间，单位为秒，默认为 1 秒。这里可以填浮点数，如 0.5 秒。

### 5. Start Delay

播放延迟时间，单位为秒，默认为 0 秒，支持浮点数。

### 6. Tween Group

动画所属的组，默认为 0。这个设置会在后文讲解 `UIPlayTween` 时讲到，它的主要作用是通过分组来整体控制多个 Tween 动画。

### 7. Ignore TimeScale

是否忽略时间缩放，默认为否。

这个选项用得极少，但是知识点很重要。我们都知道 Unity 中有一个 `TimeScale` 的概念，我们在某些比较简单的情况下会直接使用 `TimeScale=0` 这种方式来使游戏中的时间停下，达到游戏暂停的效果。而这里的是否忽略时间缩放，就是指当游戏中的 `TimeScale` 改变时，这个动画的时间是否跟着受影响。

### 8. On Finished

在动画播放完毕时候触发的函数，这里的设置和之前 UI 控件的回调函数设置方法一样，就不多描述了。

#### 4.2.3 使用透明度动画的注意点

NGUI 的 Tween 动画看上去都比较简单，容易操作。但是，在实际开发中使用透明度动画



时，我们要注意一些比较隐藏的、又比较重要的规律。

(1) 从起点播放到终点，就算动画播放完了一遍。但是，在动画结束的触发事件中，我们需要注意的是，如果动画模式是 **Loop** 或者 **PingPong**，那么它将永远不会结束。

(2) 透明度动画会实实在在地改变 UI 的透明度，并不是一个“临时”透明度。例如，我们设置了一个透明度从 1 变为 0 的渐渐消失动画，当透明度变化到 0.5 时，我们就将动画组件关闭，此时 UI 的透明度将会一直停留在 0.5。

(3) 如果有自定义动画曲线，那一定要检查曲线是否和自己想要的效果一致。

(4) 动画组件激活后，它会立即开启 **StartDelay** 的计时，然后播放动画。

(5) 如果动画播放设定为播放一次，那么动画播放一次之后，就会自动关闭该组件。

## 4.3 颜色变化动画（变色动画）

### 4.3.1 变色动画的介绍和应用

变色动画是通过 NGUI 的 **TweenColor** 组件来实时改变 UI 控件的颜色来实现动画效果。我们之前讲过 UI 控件的 **widget** 中改变 **Color** 的设置中，原理是颜色的相乘。UI 图片中的每个像素都有一个 **RGB** 值来表示这个像素的颜色，在 **Unity** 中会将 **RGB3** 个值由 0~255 的数值转变为 0~1 的一个数，然后和 UI 控件的 **widget** 的 **Color** 中设定的颜色值（设置的时候是为 **RGB** 分别填入 0~255 的数，但是也会被转换为 0~1 的数）进行相乘。在变色动画中，原理也是色值相乘。

例如，设置动画的起点是白色，终点是黑色，那么 UI 的最终颜色将会是：UI 图片的原色 \* 白色 → UI 图片的原色 \* 黑色。这个动画变化过程也是插值的。

变色动画在游戏中的应用非常广泛，比如我们选中某个 UI 时，它会明暗闪烁，则是一个白色到黑色的 **PingPong** 动画。再比如当某个 UI 我们无法单击时，它会不停地闪红色，则是一个白色到红色的 **PingPong** 动画。

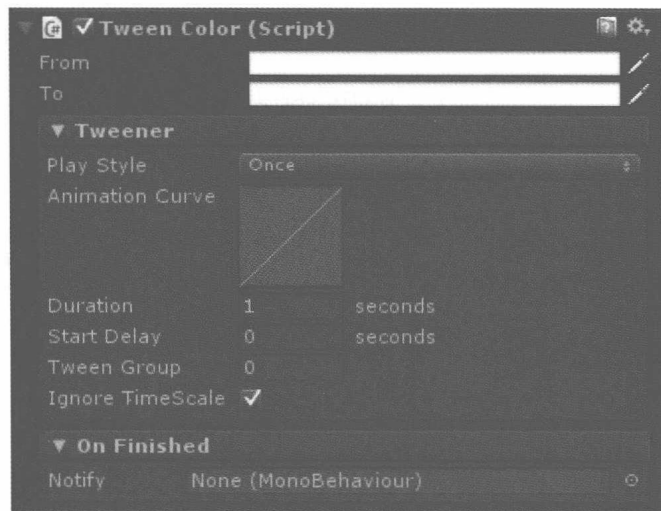
值得注意的是，我们在制作游戏的 UI 动画时，一定要区分透明度动画和颜色动画，比如说明暗变化，如果用透明度动画来制作，它其实没有变暗，是变得半透明了看上去有一点变暗的效果，而用颜色动画，则透明度没有变化，是颜色实实在在变暗了。

## 第4章 UI动画

### 4.3.2 使用颜色动画 TweenColor

我们需要为某个 UI 增加变色动画时,可以通过为它增加一个 TweenColor 组件,这个组件在 NGUI 的 Tween 动画库中,添加方式为,依次选择 AddCompent→NGUI→Tween→TweenColor。

添加成功后的 TweenColor 组件设置界面如图 4.8 所示,我们来了解一下如何设置它来制作一个变色动画。



▲图 4.8

从图 4.8 中可以看到,变色动画的组件和透明度动画的组件从界面上来看非常相似。是的,所有的 NGUI 自带的 Tween 动画,几乎都是一模一样的结构,只是其中的设置有一些不同。

TweenColor 一样只支持起点和终点两个点的设置,然后通过插值曲线来过渡。其中 Tweener 模块和之前讲过的透明度动画的 Tweener 模块是一样的,这里就不多赘述。如果对 Tweener 模块还不了解,可以参看 4.2 节。

关于这个 TweenColor 组件,我们要着重讲一下它的起始点设置。它的起始点设置是两个颜色设置板,和 UI 控件的 widget 模块中的颜色设置板一模一样。我们可以点进去设置想要的颜色。既然是用 UI 控件的 widget 模块中的颜色设置板来设置颜色,颜色有 RGBA4 个值,所以也可以通过设置它的 A 值来实现一个透明度动画。

### 4.3.3 使用颜色动画的注意点

使用颜色动画的时候,需要注意以下一些注意点。

(1) 从起点播放到终点，就算动画播放完了一遍。但是，在动画结束的触发事件中，需要注意的是，如果动画模式是 Loop 或者 PingPong，那么它将永远不会结束。

(2) 颜色动画是实实在在地改变了 UI 控件的颜色，并不是“临时改变”。例如，一个动画是将 UI 从白变黑，那么当动画播放完毕之后，UI 是真的变黑了，并不会自动变回原来的颜色。

(3) 颜色改变的原理是将 UI 控件的原色和动画中设置的颜色进行相乘，这个原理一定要深刻理解，如图 4.9 所示，则是一个动画的范例，在 TweenColor 中设置动画的起点为白色，终点为红色。



▲图 4.9

## 4.4 位置变换动画（位移动画）

### 4.4.1 位移动画的介绍和应用

NGUI 中的每一个 UI 控件其实都是一个带有 NGUI 脚本的 Unity 游戏物体 (GameObject)，它们都有一个 Transform 组件来管理这个物体的位置、旋转、缩放。位置变换动画，就是通过改变这个 UI 物体的 Transform 组件的 Position 实现的。

值得注意的是，在 NGUI 的 UIRoot 树状结构中的物体，Transform 的 Position 一般都是本地坐标（经过了 UIRoot 的缩放），而位移动画的核心组件 TweenPosition 改变的是 Position 中显示的坐标。也就是说，如果我们对 UIRoot 下的 UI 物体使用 TweenPosition，则单位可以理解为像素，如果我们自己在摄像机中挂上一个 UICamera，然后对这个 Camera 照射到的物体（不属于某个 UIRoot）使用 TweenPosition，则单位是 Unity 中的标准单位：米。

位移动画也是在开发游戏时经常使用的一种动画，比如某个按钮或者图片一直不停地上下漂浮等。再比如我们制作比较常用的“抽屉界面”都会用到位移动画，“抽屉界面”的制作我们后面会讲解。

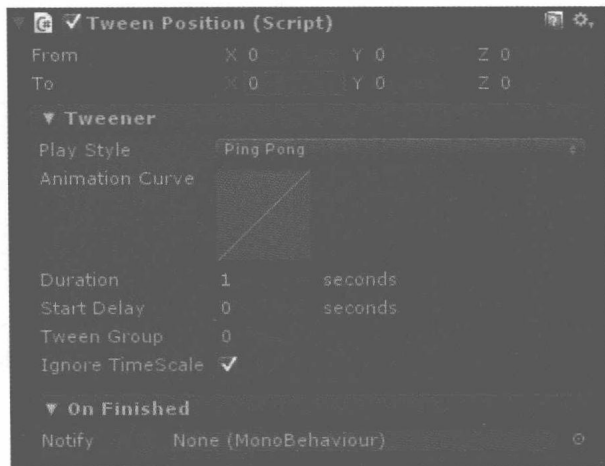


## 第4章 UI 动画

### 4.4.2 使用位移动画 TweenPosition

我们需要为某个 UI 增加位移动画时,可以通过为它增加一个 TweenPosition 组件,这个组件在 NGUI 的 Tween 动画库中,添加方式为,依次选择 AddCompent→NGUI→Tween→TweenPosition。

添加成功后的 TweenPosition 组件设置界面如图 4.10 所示,我们了解一下如何设置它来制作一个位移动画。



▲图 4.10

TweenPosition 的组件设置和其他 Tween 动画很相似,惟一的区别在于 From 和 To 是一个 Vector3 变量,起始点的设置都需要分别填入 X、Y、Z 的值。当我们成功添加了该组件之后,From 和 To 中会自动读取该物体当前的 Transform 中 Position 的值作为默认值。

### 4.4.3 使用位移动画的注意点

使用位移动画的注意点可以参看 4.2 节讲解透明度动画时的一些注意点, Tween 动画的很多注意点几乎都是一样的,比如动画如果选择了 Loop 和 PingPong,那么就不会有结束的那一刻。

除此之外,在制作位移动画时,需要特别注意的是位移的单位问题。因为如果物体属于一个 UIRoot,那么它的位置信息是相对位置,单位也是像素。如果物体不属于 UIRoot,那么它的位置信息是空间位置,单位是米。

在制作位移动画时,动画在执行时会先瞬间改变到起始点,然后开始插值位移,所以,我们一定要注意起始点的位置是否是想要的位置,一般情况下,需要起始点的位置是物体的



当前位置。虽然添加完组件之后它会自动读取当前位置信息作为默认的位置，但是，如果改变了物体的位置信息，TweenPosition 组件中的值是不会再次自动地去读取的，所以，需要经常检查 From 和 To 的值。

## 4.5 旋转变化动画（旋转动画）

### 4.5.1 旋转动画的介绍和应用

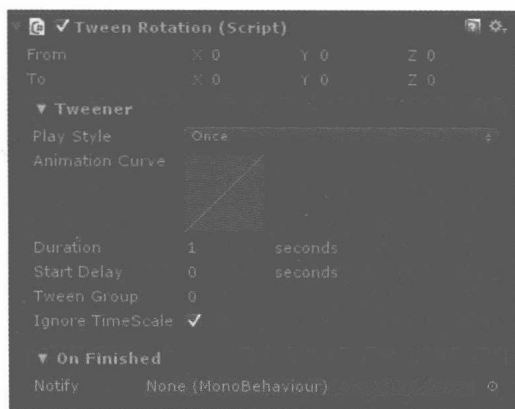
Unity 的每一个 GameObject 都有一个 Transform 组件，每一个 Transform 组件上都有 Position、Rotation、Scale 3 个信息。上文我们讲解的位移动画则是通过改变 Transform 的 Position 来实现的，这里要讲的旋转动画则是通过改变 Transform 组件的 Rotation 来实现的。

旋转动画在游戏中的应用相对要少一些，比如说通过某个操作会使 UI 图片旋转一下。

### 4.5.2 使用旋转动画 TweenRotation

我们需要为某个 UI 增加旋转动画时，可以通过为它增加一个 TweenRotation 组件，这个组件在 NGUI 的 Tween 动画库中，添加方式为，依次选择 AddComponent→NGUI→Tween→TweenRotation。

添加成功后的 TweenRotation 组件设置界面如图 4.11 所示，我们了解一下如何设置它来制作一个旋转动画。



▲图 4.11

从图 4.11 中可以看到，它的设置界面和位移动画非常相似。相同的一些设置就不多赘述了。主要来讲一下它的 From 和 To 起始点设置。

## 第4章 UI动画

旋转动画的 From 和 To 需要填入的是 3 个方向的旋转角度, 0 就是  $0^{\circ}$ , 也就是不旋转。需要注意的是,  $360^{\circ}$ , 并不等于 0 度, 因为动画有插值算法, 例如, 在 X 中填写 360, 则会顺着 X 轴的方向将图片翻转  $360^{\circ}$  一整圈。

关于旋转动画的旋转方向, 它会根据填入的 X、Y、Z 值合成一个方向 (这个 X、Y、Z 值本身就组成了 Vector3 向量), 如果希望它是朝着轴正向的, 则填入正数, 如果希望它是反向的, 则填入负数。

例如, 设置 From 的 X、Y、Z 都为 0, 也就是以当前原始姿态作为起点, 然后在 To 中的 X、Y、Z 中分别填写 X 为 60、Y 为 180、Z 为 -360。则在持续时间内, 它会插值的沿着 X 轴正向旋转  $60^{\circ}$ , 同时朝着 Y 轴正向旋转  $180^{\circ}$ , 同时朝着 Z 轴的负方向旋转  $360^{\circ}$ , 然后在同一个时间点 (动画的结束时间) 旋转结束。

我们在设置 From 和 To 时, 虽然填写的是一个旋转度数, 我们可以填写超过 360 的数字, 例如, 在 To 中填写 720, 则是单次动画过程中旋转  $720^{\circ}$ , 也就是两圈。

### 4.5.3 使用旋转动画的注意事项

使用旋转动画时, 我们要注意以下几点。

(1) 动画的方向不要弄错, 它是由 X、Y、Z 上的 3 个方向叠加起来的。如果只需要 UI 朝一个方向旋转, 请将另外两个方向的值设为 0。

(2) 只有 From 的 X、Y、Z 都为 0 时, 才是从原始形态开始旋转。

(3)  $0^{\circ}$  和  $360^{\circ}$  区别很大。虽然  $0^{\circ}$  和  $360^{\circ}$  看到的结果是一样的, 但是, 因为动画有一个插值过渡的过程, 所以  $0^{\circ}$  不会旋转, 而  $360^{\circ}$  则会旋转一整圈回到原样。

(4) 记住正数为正方向, 负数为负方向。

(5) 填写的数值可以超过 360, 例如, 填写一个  $720^{\circ}$  旋转, 则是单次动画转两圈的意思。

## 4.6 大小变化动画 (放缩动画)

### 4.6.1 放缩动画的介绍和应用

放缩动画和旋转动画和位移动画一样, 都是通过改变物体的 Transform 组件来实现的。位移动画是改变的 Transform 的 Position 信息, 旋转动画改变的是 Transform 的 Rotation 信息,