

Unity 2D 游戏开发快速入门

——《狂怒坦克 RAGETANK》制作

（内部资料）



大学霸

www.daxueba.net

前言

Unity 是一款综合的游戏开发工具，也是一款全面整合的专业游戏引擎。使用它开发的游戏，可以部署到所有的主流游戏平台，而无需做任何修改。这样，开发者只需把精力集中到制作高质量的游戏即可。

本书通篇介绍了一个 2D 游戏——RageTanks（狂怒坦克）的详细开发过程，包从导入游戏资源、游戏逻辑设计到最后游戏逻辑的实现。本书将这一开发过程分成了 6 个部分来讲解，力求读者在每一部分都能实现一个可见的效果，而这些效果的综合体现就是最后的 RageTanks。

没用过 Unity？没关系，这里有详细的操作步骤；

没学过 C#？没关系，这里有详细的注释和解释，更何况 C# 本来就不难；

得学习很久吗？不，即时你是新手，依然可以在一个月内做出本书介绍的这个游戏！

喔~说的夸张吗？一点儿也不！为什么这么自信，因为我是作者！我精心设计了这个游戏！它简单、结构清晰，而且也很有趣！我相信你通过对这个游戏的学习，可以增进对 3 个方面的理解：Unity、2D 游戏开发流程和脚本代码的编写。

1. 学习所需的系统和软件

- ☐ 安装 Windows 7 操作系统
- ☐ 安装 Unity 4.5.3

2. 学习建议

大家学习之前，可以致信到 xxxxxxx，获取相关的资料 and 软件。如果大家在学习过程遇到问题，也可以将问题发送到该邮箱。我们尽可能给大家解决。

目 录

第 1 章 创建一个简单的 2D 游戏	1
1.1 地面	1
1.2 游戏精灵	3
1.3 精灵动画	7
1.3.1 Animation	7
1.3.2 Animator	9
1.4 使用脚本实现游戏逻辑	12
精灵动画状态的控制	12
监听精灵当前的动画状态	14
1.5 2D 游戏的运行效果	17
第 2 章 为游戏精灵添加更多状态	19
2.1 摄像头追踪功能	19
2.2 精灵的死亡和重生	22
2.3 添加多个地面	27
2.4 精灵的跳跃状态	28
2.5 精灵的开火状态	34
第 3 章 让游戏精灵不再孤单	40
3.1 为游戏添加反派角色	40
3.2 精灵与反派角色碰撞后死亡	44
3.3 精灵主动攻击反派角色	46
3.4 添加反派角色销毁时的效果	48
3.5 添加多个反派角色到游戏中	50
第 4 章 为游戏添加更多背景元素	52
4.1 为游戏场景补充更多元素	52
4.1.1 限制精灵的移动范围	52
4.1.2 添加背景元素	54
4.1.3 让背景元素动起来	55
4.1.4 让粒子效果显示在前面	58
4.2 记录分数	59
4.3 动态生成更多的敌人	61
第 5 章 终极战斗	66
5.1 引入究极敌人	66
5.2 究极敌人的行为逻辑	67
5.3 让究极敌人的出场更威风些	72
5.4 究极敌人的攻击方式	74
5.5 玩家精灵的反击	77

第 6 章 让游戏更完善	85
6.1 游戏关卡	85
6.2 游戏标题以及开始按钮	88
6.2.1 导入标题和按钮资源	88
6.2.2 表示游戏状态的类	89
6.2.3 单击开始按钮，进入游戏	92
6.2.4 游戏最终运行效果展示	94



第 1 章 创建一个简单的 2D 游戏

即使是现在，很多初学游戏开发的同学，在谈到 Unity 的时候，依然会认为 Unity 只能用于制作 3D 游戏的。实际上，Unity 在 2013 年发布 4.3 版本的时候，就开始提供对制作 2D 游戏的支持了。例如，提供了一些专用于开发 2D 游戏的 Unity 工具。现在 Unity 已经发布了版本 4.5，对 2D 游戏的支持更是完善了不少。为了说明 Unity 对 2D 游戏所提供的支持，本章会使用这些在 Unity 中原生的工具，开发一个简单的 2D 游戏。

1.1 地面

一般情况下我们认为“地面”是精灵的落脚点，没有这个“地面”精灵会发生坠落。因此本章创建一个 2D 游戏的第一步是，创建一个供游戏精灵落脚的“地面”。创建的具体的操作步骤是：

（1）新建一个游戏项目、命名、并设置为 2D 项目，如图 1-1 所示。从中可以看出本游戏项目名为 world。

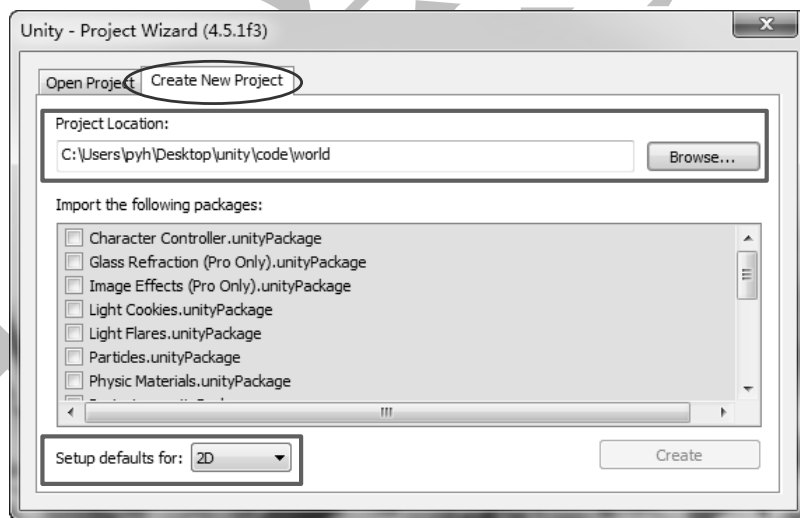


图 1-1 新建游戏项目

（2）准备一个将用作 2D 游戏“地面”的图片，然后导入到新建游戏项目的 Project 视图里，如图 1-2 所示，本示例使用的“地面”图片，名为 Platform。

提示 1：导入资源到游戏项目的简单的方法是，直接拖动对应资源到 Project 视图下。此种方法可以一次拖入一个或者多个资源。

提示 2：制作游戏的过程中，会不断的导入游戏所需的资源，如果不提前做出整理规划的话，导入的资源多了以后会十分的“乱”，不仅看起来不舒服，找起来也会十分的麻烦。因此，最好养成一个资源分类存放的好习惯。在本示例中，导入的“地面”图片，被放置在了 Assets\Textures\Scenery 文件夹里。

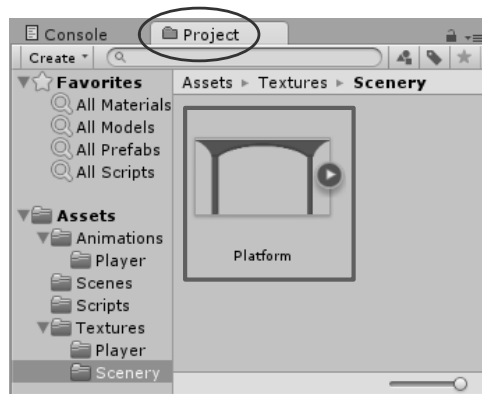


图 1-2 Unity 中，Project 视图里，是导入的图片

(3) 选中 Project 视图里的 Platform，然后 Inspector 视图会显示这个图片的各项属性，以及此图片的预览视图，如图 1-3 所示。

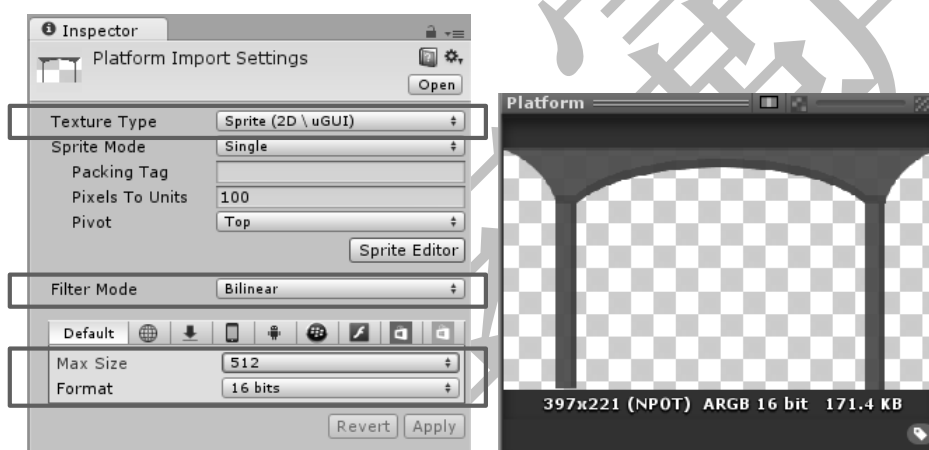


图 1-3 Inspector 视图里显示了 Platform 的各项属性

对于此图片，有以下几个属性需要简单说明：

- ❑ **Texture Type:** 表示图片的类型，不同的类型在项目里的用途就不同。对于 2D 游戏项目，系统默认设置为 Sprite(2D\uGUI)；对于 3D 游戏项目，默认设置为 Texture；
- ❑ **Filter Mode:** 此属性一共有 3 个选项：Point、Bilinear 和 Trilinear。设置了以后，图片用在游戏中时，质量或者说画面效果依次升高。默认设置为 Bilinear。
- ❑ **Max Size:** 表示图片的最大尺寸。游戏中，有时会以不同的尺寸使用图片，并进行相应的伸缩。本示例使用的图片 Platform 的实际尺寸是 397×221，为了不影响图片的质量，要设置 Max Size 属性大于 397，而属性中最接近 397 的是 512。
- ❑ **Format:** 表示图片的格式。设置为 16 bits，表示此图片不是压缩图片，而且可以表示数以“万亿” ($2^{16} \times 2^{16} \times 2^{16}$) 计的颜色。

(4) 要在游戏中使用此图片的话，可以直接将此图片拖动到 Scene 或者 Hierarchy 视图里。Unity 就会在当前的游戏场景中添加这个图片对象，选中游戏场景里的此图片对象，然后在 Inspector 视图里设置它 Transform 组件下的各属性，如图 1-4 所示，

- ❑ **Position:** (0,0,0);
- ❑ **Rotation:** (0,0,0);
- ❑ **Scale:** (1,1,1);

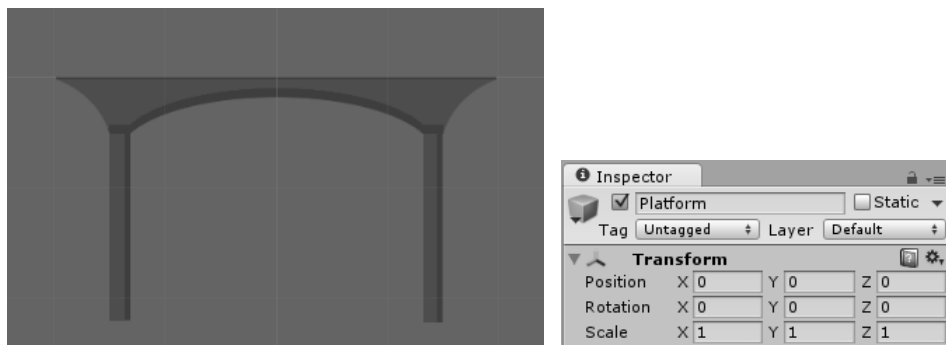


图 1-4 添加到游戏场景的图片对象，及其属性设置

（5）“地面”对象已经添加到了游戏的场景视图中，但是它的上面还不能“站立”任何精灵，也就是说它会如空气一般的存在，站立在其上面的任何精灵，都会如踩到了空气一样“下落”。所以需要给“地面”对象添加 Collider 2D 组件，一共有 4 类此种组件，如图 1-5 所示。不同的类型适合于不同外形形状的对象，由于“地面”对象的外形是不规则的，因此本示例选择使用 Polygon Collider 2D 组件。

为“地面”对象添加此组件的方法是，选中场景中的此对象，然后单击 Component|Physics 2D|Polygon Collider 2D 命令即可。添加了这个组件以后，在看场景中的“地面”对象，会发现它的表面“覆盖”了绿色的线，这个线包住了对象，如图 1-6 所示。

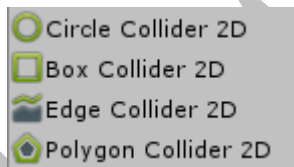


图 1-5 4 类 Collider 2D 组件

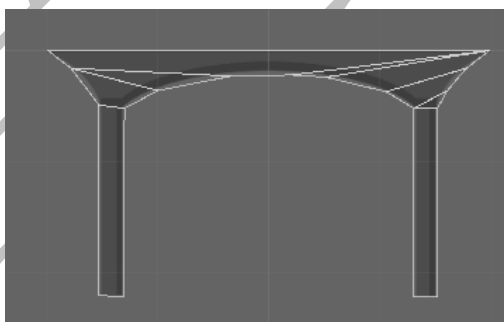


图 1-6 添加了 Polygon Collider 2D 组件的效果

1.2 游戏精灵

“地面”在游戏中已经有了，并且也设置好了，现在是时候添加游戏精灵了。与地面类似，游戏精灵也是一张图片，本示例导入的精灵图片名为 **playerSpritesheet**，如图 1-7 所示。

提示：为了 Project 视图的整洁，最好将资源分类整理，本示例将此图片置于 Assets\Textures\Player 文件夹下。

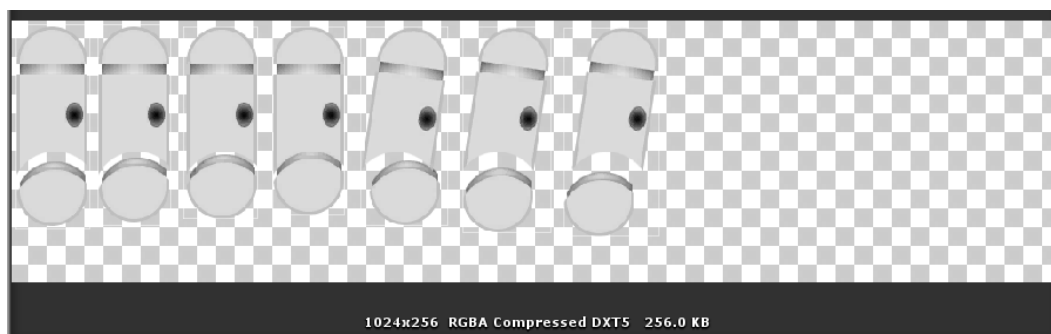


图 1-7 精灵图片

我们发现精灵图片里有多个精灵，实际上此图片也被称为“精灵图集”（sprite atlas），每一张图都表示了精灵的不同运动状态，如果将它们组合起来播放的话，精灵就会动起来了。实际上这也是本节要实现的效果。具体的操作方法是：

（1）虽然我们从图片中直接看出来这是个精灵图集，但是 Unity 并不知道。为了将这一点告知 Unity，需要选中此精灵图集，然后在 Inspector 视图里，设置其 Sprite Mode 属性为 Multiple，如图 1-8 所示。

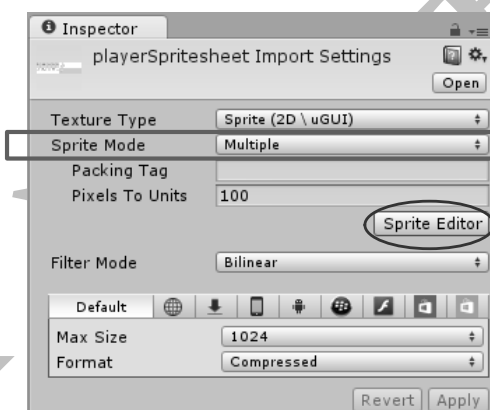


图 1-8 修改精灵图集的 Sprite Mode 属性

（2）很好，现在 Unity 知道这个图片是精灵图集了。但它还不知道里面有几个精灵，更别说精灵的其它属性了，为此我们还需要指出每个精灵。具体的实现方法是，单击 Inspector 视图里的 Sprite Editor 按钮，会打开一个名为 Sprite Editor 的窗口，在这个窗口里，使用鼠标左键拖拽的方式，可以框选每一个精灵，Unity 会认为框选住的就是具体的一个精灵。此时还可以设置此框选精灵的一些其它属性，这些属性就被显示在 Sprite Editor 视图的右下角，如图 1-9 所示。

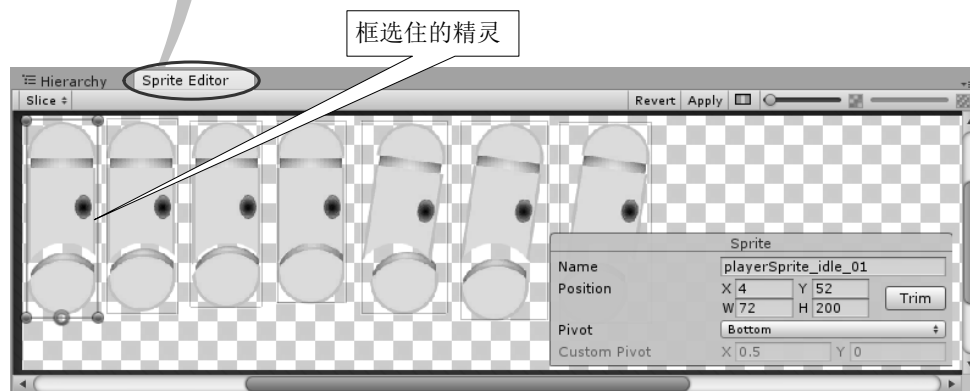


图 1-9 在 Sprite Editor 视图里框选每一个精灵，然后在视图的右下角设置其属性

(3) 单击属性设置框里的 Trim 按钮，Unity 会自动对框中的精灵做处理，即切掉没用的“空白”，得到最小的矩形框，读者可以在框选中一个精灵以后单击此按钮。每个精灵都有一个中心点，或者称为轴点（Pivot），精灵旋转时会以此轴线为中心旋转，如图 1-10 所示，此点可以位于任何位置，但是本节示例将其设置为底部的中点。

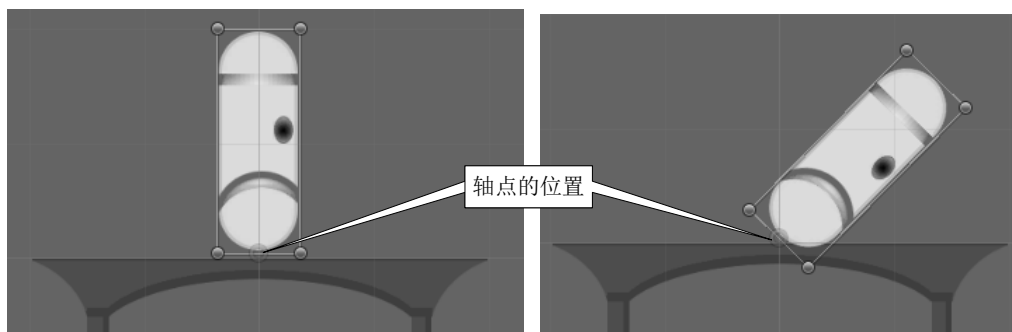


图 1-10 精灵的轴点

框选出精灵图集中的所有精灵以后，还需要为各精灵命名，本示例中将它们从左到右依次命名为：playerSprite_idle_01、playerSprite_idle_02、playerSprite_idle_03、playerSprite_idle_04、playerSprite_walk_01、playerSprite_walk_02、playerSprite_walk_03，最后，单击 Sprite Editor 视图右上角的 Apply 按钮，即可应用在 Sprite Editor 视图里所做的所有设置。

提示：本示例中的精灵图集里只有 7 个精灵，因此一个个框选的话虽然麻烦，但是还不至于绝望，如果遇到精灵图集里有上百个的精灵的话，使用这种手工的方式就太不明智了，实际上在 Sprite Editor 视图的左上角，有个名为 Slice 的按钮，它可以有效的解决这个问题。单击 Slice 按钮，会弹出一个设置框，如图 1-11 所示，读者可以自行探索尝试。

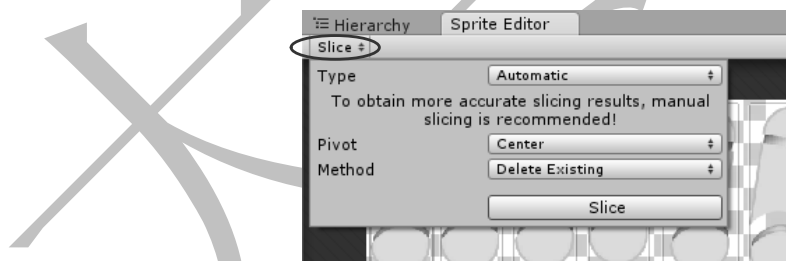


图 1-11 使用 Sprite Editor 视图左上角的 Slice 按钮，可以自动框选出精灵图集中的所有精灵

(4) 再次定位到 Project 视图里的精灵图集，会发现 Unity 按照我们之前所做的设置，找到了精灵图集中的所有精灵，且使用了我们指定的名称表示各精灵图集，如图 1-12 所示。

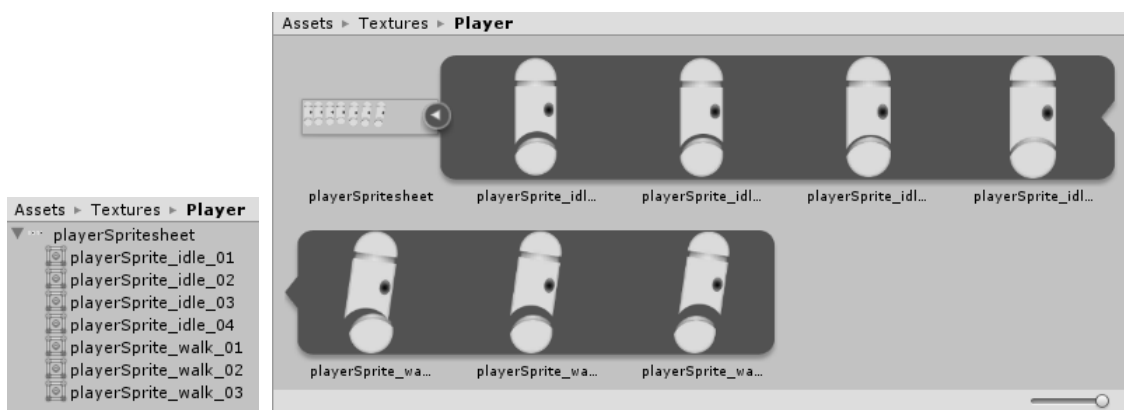


图 1-12 Unity 准确识别了精灵图集内的所有精灵

(5) 选中 Project 视图下的 `playerSprite_idle_01`，然后拖动到 Scene 视图里，即可在当前的游戏场景中添加单个精灵对象，在 Hierarchy 视图里，重命名此精灵对象为 `Player`。选中它，然后在 Inspector 视图里，设置其 Transform 组件下的属性，最后得到的游戏场景视图，如图 1-13 所示。

- ☐ Position: (0,0,0);
- ☐ Rotation: (0,0,0);
- ☐ Scale: (0.7,0.7,0);

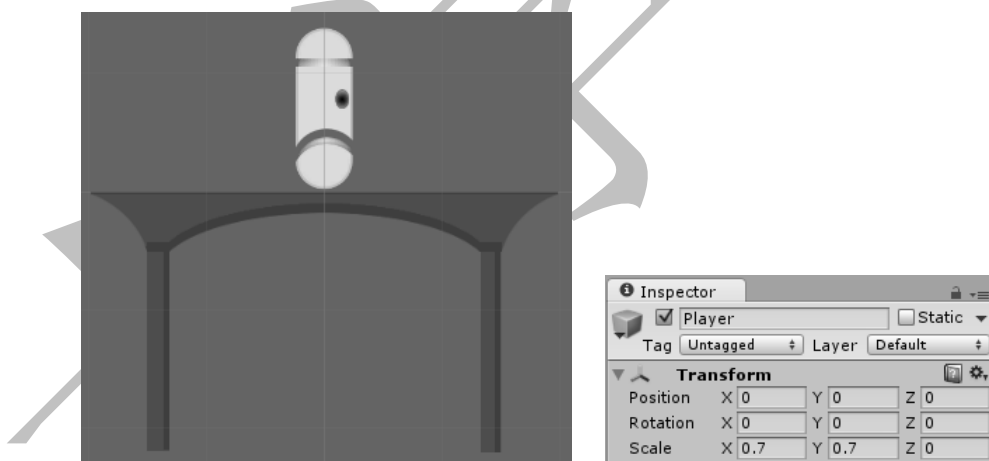


图 1-13 精灵对象的属性设置，以及当前的游戏场景视图

(6) 选中 Hierarchy 视图里的 `Player` 对象，然后单击 Component|Physics 2D|Rigidbody 2D 和 Component|Physics 2D|Polygon Collider 2D 命令，为对象添加 Rigidbody 2D 和 Polygon Collider 2D 组件，如图 1-14 所示。



图 1-14 添加到对象上的两个组件

提示：为对象添加组件还可以单击 Inspector 视图里的 Add Component 按钮，如图 1-15 所示。

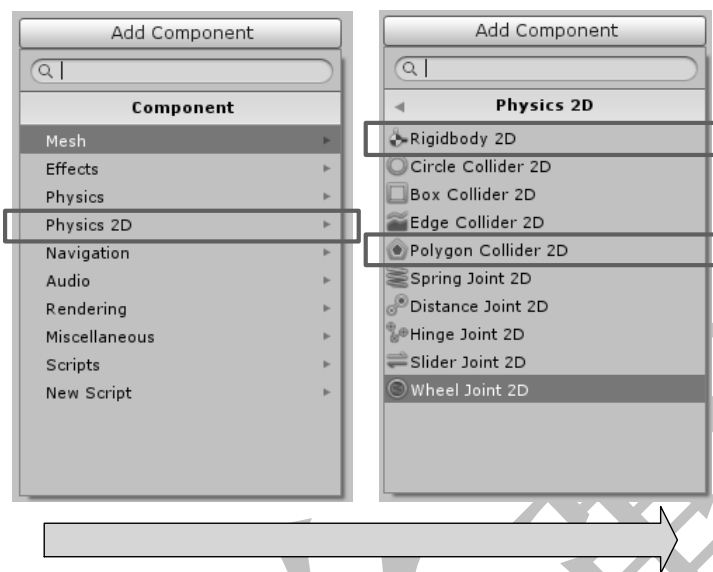


图 1-15 添加组件的另一种方式

1.3 精灵动画

场景中已经添加了精灵，现在是时候让它动起来了。读者也许已经从精灵图集中，各精灵的命名中看出来，这个精灵一共有两种动画状态：Idle（空闲）和 Walking（走）。本节将讲解使用 Unity 自带的工具，创建精灵动画，并实现简单动画控制的方法。

1.3.1 Animation

要将精灵图集集中的多个精灵的动作，组合成一个动画，可以使用 Unity 提供的 Animation 工具。具体的说明和操作方法如下：

（1）在 Unity 中，单击 Window|Animation 命令，即可打开名为 Animation 的窗口，如图 1-16 所示，在此窗口中就可以将多个精灵的动作，组合起来播放，最终形成动画。

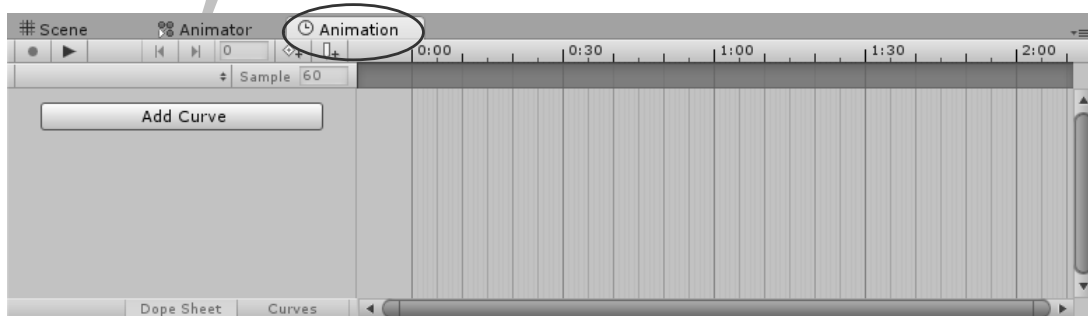



图 1-16 Animation 窗口

(2) 选中 Hierarchy 视图里的 Player 对象（即游戏场景中的精灵对象，现在要为其添加动画效果），然后进入到 Animation 视图里。单击视图左上角的  按钮，会弹出一个名为 Create New Animation 的对话框，如图 1-17 所示。此时需要指定要制作的动画的名字和存储位置。

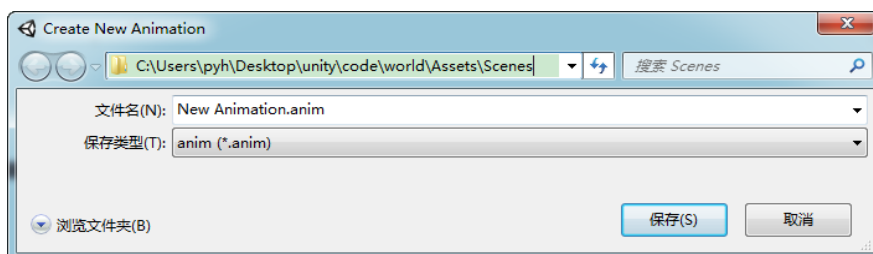


图 1-17 Create New Animation 对话框

提示：在本示例中，将此动画命名为 `PlayerIdleAnimation.anim`，并存储于 `Assets\Animations\Player` 文件夹下。

在创建了动画以后，进入到 Project 视图里，创建了动画的指定路径里，我们会发现里面不光创建了指定名称的动画文件，还创建了一个名为 `Player` 的动画控制器（Animator Controller）。因为我们选中了 Hierarchy 视图里的 Player 对象，所以才新建了同名的动画控制器。修改它的名字为 `PlayerAnimatorController`，这样的命名对它来说更准确些。

提示：动画控制器可以控制精灵对象，做出各种动画效果，并且还可以设置各种动画的过渡条件等等，关于这些会在下面的各步骤中讲解。

(3) 新建了动画以后，再选中 Hierarchy 视图里的 Player 对象，然后在 Inspector 视图里可以看到，此对象里多了一个组件 Animator，且其属性 Controller 已经设置成了新建的动画控制器，即 `PlayerAnimatorController`，如图 1-18 所示。

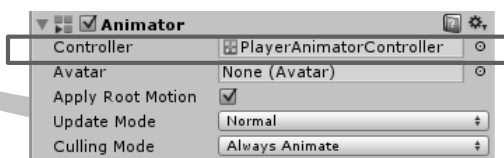


图 1-18 Player 对象上，新添加的 Animator 组件

Animator 组件借助 Controller 属性里设置的动画控制器，真正意义上完成精灵动画的播放，以及各种动画间的转换。

(4) 在选中 Hierarchy 视图里 Player 对象的前提下，进入到 Animation 视图里，会发现此视图已经自动打开了 `PlayerIdleAnimation` 动画。确保此视图左下角的 Dope Sheet 按钮处于选中状态，然后将表示同一动画的各精灵，按照动画应有的顺序依次拖入到 Animation 视图的右侧，且相互之间隔开一定的“距离”，在这里使用“动画帧”来描述或许更准确些，如图 1-19 所示。

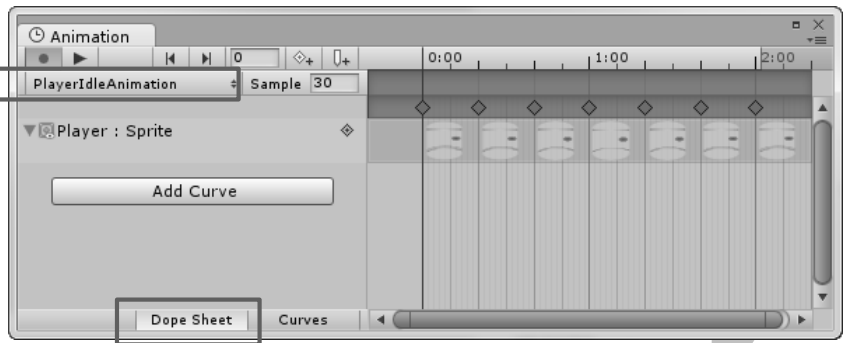



图 1-19 将精灵拖入到 Animation 视图右侧，并且相隔一定的动画帧

提示：在精灵图集中，表示 Idle 动画状态的精灵有 4 个，即 `playerSprite_idle_01~playerSprite_idle_04`，将它们依次拖动到 Animation 视图的右侧。明明只有 4 个精灵，那么为什么本示例中拖入了 7 个呢？答案是为了让动画流畅播放，即精灵的最后一个动作，应该与第一个动作一致才行，因此后三个精灵依次是 `playerSprite_idle_03~playerSprite_idle_01`。也就是将下图 1-20 所示的各精灵组合起来播放，就会形成流畅的 Idle 动画效果。



图 1-20 组成 Idle 动画的各精灵，首尾精灵动作一致

(5) 单击 Animation 视图左上角的  按钮，动画的效果就会在 Scene 视图里播放出来。如果觉得动画播放的速度太慢，可以调节 Animation 视图左侧上部 Sample（采样率）属性的值，如图 1-21 所示，它的值越大，动画的播放速度越快。

(6) 要为 Player 对象添加其它的动画（例如，本示例中的 Walking 动画），可以单击 Animation 视图左上角的动画名按钮（在本示例中其名为 `PlayerIdleAnimation`），然后选中其中的 `[Create New Clip]` 选项即可，如图 1-22 所示。

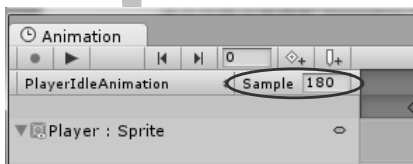


图 1-21 在 Animation 视图里，设置动画播放时的采样率

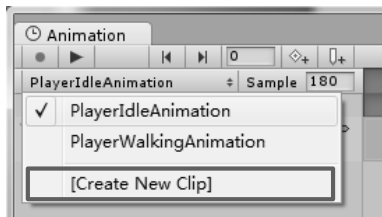



图 1-22 为 Player 对象添加其它的动画

提示：还需要为动画命名，以及指定存储位置，在本示例中动画名为 `PlayerWalkingAnimation`，存储于 `Assets\Animations\Player` 文件夹下。

接着就可以使用同样的方式，为 Player 对象添加 Walking 动画效果了。

1.3.2 Animator

通过上一小节的操作，我们新建了 2 个动画：PlayerIdleAnimation 和 PlayerWalkingAnimation。而且在单击 Animation 视图里的  按钮时，也看到了动画的效果。但是现在还需要管理这两个动画，也就是说需要能够指定精灵当前进入的动画状态，为此就需要对动画控制器展开一些设置和操作，具体的步骤是：

（1）在 Project 视图里，找到在创建动画时一并生成的动画控制器，在本示例中它的名字是 PlayerAnimatorController。双击它，然后会弹出 Animator 窗口视图，此视图包含 3 个动画状态，名为 Any State 的动画状态是系统生成的，其它两个动画状态是我们在上一小节里定义的，如图 1-23 所示。

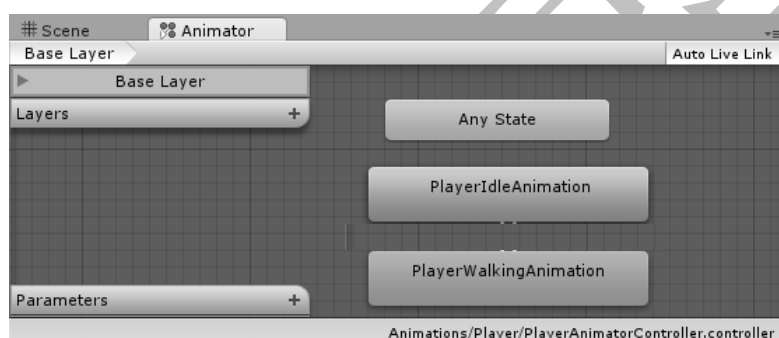


图 1-23 Animator 视图里的 3 个动画状态

（2）在 PlayerIdleAnimation 动画状态上右击鼠标，然后在弹出的快捷菜单中选择 Make Transition，最后再单击 PlayerWalkingAnimation 动画状态。这样就添加一条从前者指向后者的箭头，此箭头表示两个动画状态的过渡，即从前者表示的动画状态过渡的后者表示的动画状态。使用同样的方式，添加从后者到前后的过渡，如图 1-24 所示。



图 1-24 建立两个动画状态的过渡

（3）单击 Unity 工具栏里的开始游戏按钮，然后查看 Game 和 Animator 视图，你会发现精灵在反复的播放这两个动画状态，如图 1-25 所示，但是我们目前还无法指定其进入哪种动画状态。

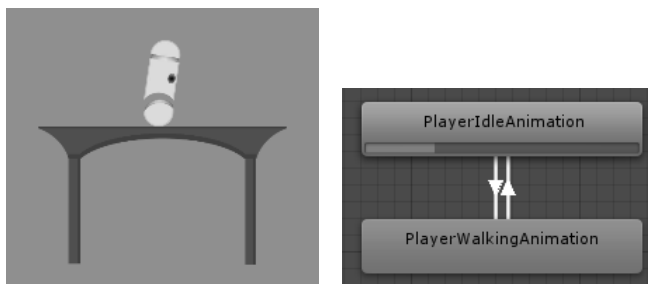


图 1-25 Game 和 Animator 视图

（4）要想指定当前精灵所进入的动画状态，就需要设置两个动画状态的过渡条件。在 Animator 视图的左下角有个名为 Parameters 的小窗口，单击其右侧的 + 按钮，可以添加参数。在本示例中，需要添加名为 Walking 的 Bool 类型的参数，如图 1-26 所示。

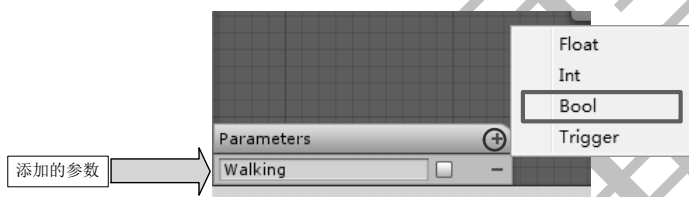


图 1-26 在 Animator 视图里，添加一个 Bool 类型的参数

（5）然后在 Animator 视图里，单击表示动画状态从 PlayerIdleAnimation 过渡到 PlayerWalkingAnimation 的箭头（箭头的颜色由白色变为蓝色），再查看 Inspector 视图，如图 1-27 所示，在视图靠下面的地方有个 Conditions，它用于设置动画过渡的条件，此时系统所设置的动画过渡条件是，在动画播放 1 秒后，进入另一个动画状态，因此之前我们播放动画的时候发现，精灵一直在两种动画状态间跳转。

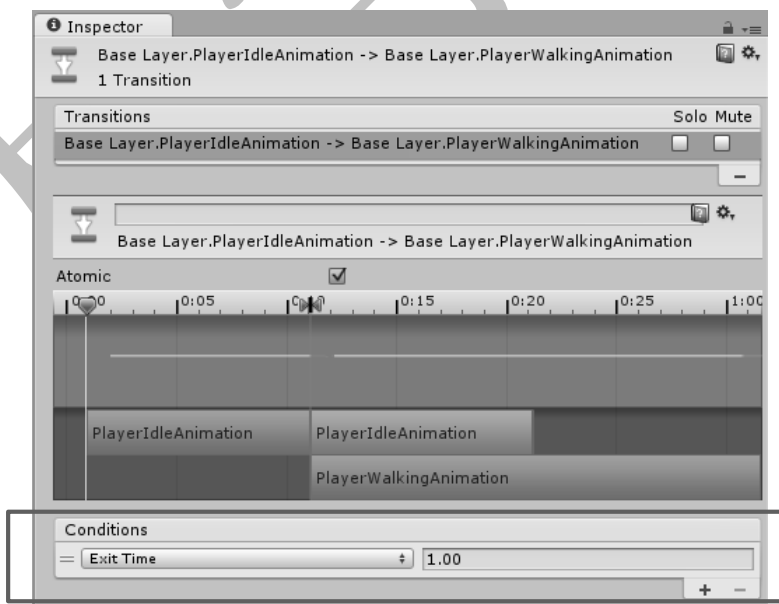


图 1-27 Inspector 视图里，动画状态的过渡属性设置

（6）单击 Condition 下的 Exit Time，从中选中我们刚才定义的参数 Walking，然后设置其后面的属性为 true，如图 1-28 所示。表示只有在参数 Walking 为 true 时，动画状态才会从 PlayerIdleAnimation 过渡到 PlayerWalkingAnimation。

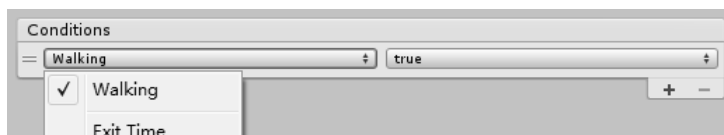


图 1-28 设置动画状态的过渡条件

使用同样的方式设置，从 `PlayerWalkingAnimation` 到 `PlayerIdleAnimation` 的过渡条件为 `Walking`，设置其属性为 `false`，表示 `Walk` 为 `false` 时，动画状态的过渡才会发生。

(7) 运行游戏，然后查看 `Game` 和 `Animator` 视图，如图 1-29 所示。默认情况下，`Walking` 的值为 `false`，因此 `Game` 视图里，一直在播放 `PlayerIdleAnimator` 动画。

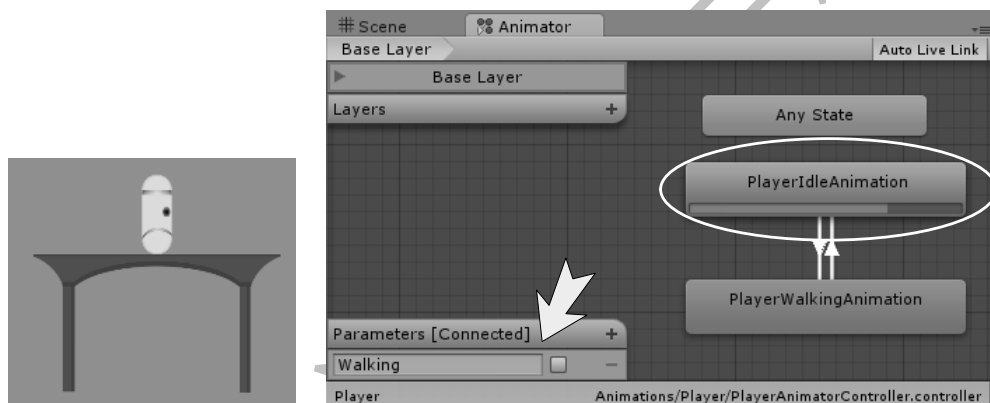


图 1-29 Game 和 Animator 视图

单击 `Animator` 视图左下角 `Parameters` 下的 `Walking` 参数的复选框，即设置 `Walking` 为 `true`，再次查看 `Animator` 视图，此时 `Game` 视图中的精灵则是一直在播放 `PlayerWalkingAnimation` 动画，如图 1-30 所示。

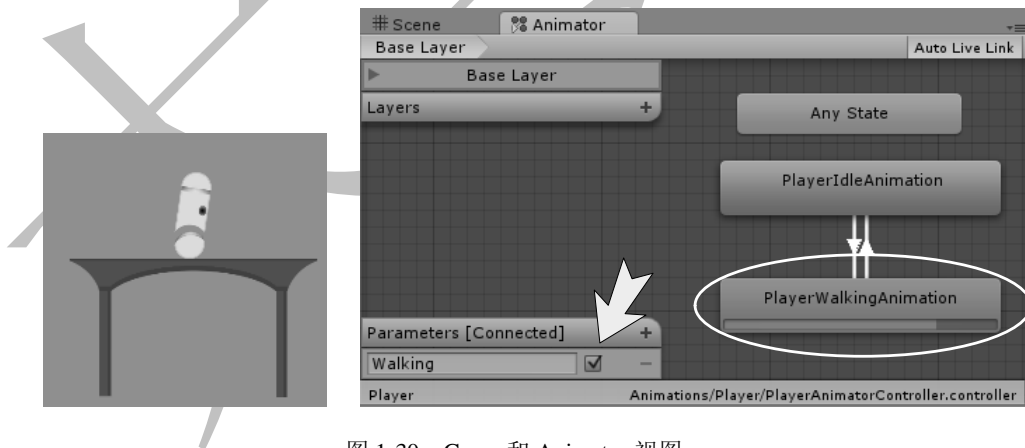


图 1-30 Game 和 Animator 视图

1.4 使用脚本实现游戏逻辑

通过上一节的操作，我们不仅创建了精灵的动画，还设置了动画的过渡条件，最终使得精灵得以按照我们的意愿，进入我们所指定的动画状态。但是这其中还有一些问题。例如，我们无法使用键盘控制精灵当前要进入的动画状态，而且精灵也只是在原地播放动画而已。

但我们希望精灵在进入 `PlayerWalkingAnimation` 状态时，位置应该发生改变。

要解决这些问题，就需要编写脚本。也就是说，要使用脚本来实现动画的播放控制，以及其它一些游戏的逻辑。

精灵动画状态的控制

在 `Project` 视图里，新建一个文件夹，命名为 `Scripts`，在此文件夹里新建一个 C# 脚本，命名为 `PlayerStateController`，然后为此脚本添加下面的代码：

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerStateController : MonoBehaviour
05 {
06     //定义游戏人物的状态
07     public enum playerStates
08     {
09         idle = 0,                //表示空闲
10         left,                    //表示左移
11         right,                   //表示右移
12     }
13     //定义委托和事件
14     public delegate void playerStateHandler(PlayerStateController.playerStates newState);
15     public static event playerStateHandler onStateChange;
16     void LateUpdate ()
17     {
18         //获取玩家在键盘上对 A、D，或者左、右方向键的输入
19         float horizontal = Input.GetAxis("Horizontal");
20         if(horizontal != 0.0f)
21         {
22             //如果按下的是左方向键，则广播左移状态
23             if(horizontal < 0.0f)
24             {
25                 if(onStateChange != null)
26                     onStateChange(PlayerStateController.playerStates.left);
27             }
28             //如果按下的是右方向键，则广播右移状态
29             else
30             {
31                 if(onStateChange != null)
32                     onStateChange(PlayerStateController.playerStates.right);
33             }
34         }
35         else
36         {
37             //广播空闲状态
38             if(onStateChange != null)
39                 onStateChange(PlayerStateController.playerStates.idle);
40         }
41     }
42 }
```

将此脚本赋予 Hierarchy 视图里的 Player 对象。对于此脚本，有以下几点需要说明：

- ❑ 脚本 07 行，定义了名为 `playerStates` 的枚举类型，此类型中定义了精灵的具体状态；
- ❑ 脚本 19 行，会获取玩家在键盘上对指定按键的输入。不同的输入值会直接导致精灵进入不同的游戏状态；
- ❑ 此脚本中定义的是一个可以控制精灵当前所进入动画状态的类，也可以认为是一个管理类型的类，只负责向精灵“发号施令”，具体的实现过程不是这个类要考虑的；

监听精灵当前的动画状态

上一小节里，我们使用脚本，定义了一个用于“发号施令”的类，而本小节将使用脚本定义一个“负责具体执行”的类。

在 Project 视图里的 Scripts 文件夹下，新建一个 C# 脚本，命名为 `PlayerStateListener`，为此脚本添加下面的代码：

```
01 using UnityEngine;
02 using System.Collections;
03
04 [RequireComponent(typeof(Animator))]
05 public class PlayerStateListener : MonoBehaviour
06 {
07     public float playerWalkSpeed = 3f;           //表示精灵移动的速度
08     private Animator playerAnimator = null;       //表示对象上的 Animator 组件
09     //表示精灵当前的动画状态
10     private PlayerStateController.playerStates currentState =
PlayerStateController.playerStates.idle;
11     //对象可用时，加入到订阅者列表中
12     void OnEnable()
13     {
14         PlayerStateController.onStateChange += onStateChange;
15     }
16     //不可用时，从订阅者列表中退出
17     void OnDisable()
18     {
19         PlayerStateController.onStateChange -= onStateChange;
20     }
21     void Start()
22     {
23         //建立与对象上 Animator 组件的联系
24         playerAnimator = GetComponent<Animator>();
25     }
26     void LateUpdate()
27     {
28         onStateCycle();
29     }
30     //用于检测当前所处的动画状态，在不同的状态下将表现出不同的行为
31     void onStateCycle()
32     {
33         //表示当前对象的大小
```

```
34     Vector3 localScale = transform.localScale;
35     //判断当前处于何种状态
36     switch(currentState)
37     {
38         case PlayerStateController.playerStates.idle:
39             break;
40         //向左移动
41         case PlayerStateController.playerStates.left:
42             transform.Translate(
43                 new Vector3((playerWalkSpeed * -1.0f) * Time.deltaTime, 0.0f,
44 0.0f));
45             //角色将转向
46             if(localScale.x > 0.0f)
47             {
48                 localScale.x *= -1.0f;
49                 transform.localScale = localScale;
50             }
51             break;
52         //向右移动
53         case PlayerStateController.playerStates.right:
54             transform.Translate(new Vector3(playerWalkSpeed * Time.deltaTime,
55 0.0f, 0.0f));
56             //角色将转向
57             if(localScale.x < 0.0f)
58             {
59                 localScale.x *= -1.0f;
60                 transform.localScale = localScale;
61             }
62             break;
63     }
64     //当角色的状态发生改变的时候，调用此函数
65     public void onStateChange(PlayerStateController.playerStates newState)
66     {
67         //如果状态没有发生变化，则无需改变状态
68         if(newState == currentState)
69             return;
70         //判断精灵能否由当前的动画状态，直接转换为另一个动画状态
71         if(!checkForValidStatePair(newState))
72             return;
73         //通过修改 Parameter 中 Walking 的值，修改精灵当前的状态
74         switch(newState)
75         {
76             case PlayerStateController.playerStates.idle:
77                 playerAnimator.SetBool("Walking", false);
78             break;
79             case PlayerStateController.playerStates.left:
80                 playerAnimator.SetBool("Walking", true);
81             break;
82             case PlayerStateController.playerStates.right:
83                 playerAnimator.SetBool("Walking", true);
```

```

84         break;
85     }
86     //记录角色当前的状态
87     currentState = newState;
88 }
89
90 //用于确认当前的动画状态能否直接转换为另一动画状态的函数
91 bool checkForValidStatePair(PlayerStateController.playerStates newState)
92 {
93     bool returnVal = false;
94     //比较两种动画状态
95     switch(currentState)
96     {
97         case PlayerStateController.playerStates.idle:
98             returnVal = true;
99             break;
100        case PlayerStateController.playerStates.left:
101            returnVal = true;
102            break;
103        case PlayerStateController.playerStates.right:
104            returnVal = true;
105            break;
106    }
107    return returnVal;
108 }
109 }

```

将此脚本赋予 Hierarchy 视图里的 Player 对象，选中后者，然后在 Inspector 视图里，找到此脚本组件，发现里面有一个属性 Player Walk Speed，如图 1-31 所示。正如属性名的含义，它可以用于设置精灵的移动速度，并且值越大，精灵的移动速度越快。

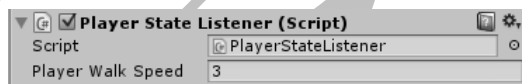


图 1-31 脚本组件，及其属性

对于此脚本，有以下几点需要说明：

- ❑ 脚本 10 行，说明默认情况下，精灵在游戏开始时，所处的是 Idle 动画状态；
- ❑ 脚本 12、17 行定义的方法 OnEnable()和 OnDisable()，说明只有成为订阅者的对象才会收到动画状态改变的消息；
- ❑ 脚本 31 行定义的方法 onStateCycle()，可以依据精灵当前所处的动画状态，修改精灵在场景中的位置属性。例如，当精灵进入左移的动画状态时，就向左修改场景里精灵的位置；
- ❑ 脚本 65 行定义的方法 onStateChange()，用于真正修改精灵当前所处的动画状态，也就是通过设置 Parameters 中 Walking 的值，进而完成的精灵动画状态的切换；
- ❑ 脚本 91 行定义的方法 checkForValidStatePair()，说明动画状态并非可以随意转换，这是为了更加符合逻辑。例如，人可以从“跑”的状态过渡到“跑跳”的状态，但是无法从“走”的状态过渡到“跑跳”的状态。
- ❑ 脚本中 44~49、54~59 行的代码应该被关注，因为它使得精灵实现了“转

身”。为什么这么说呢？因为没有它们的话，精灵会始终朝向一个方向，读者可以注释掉它们并运行程序查看效果。精灵默认在游戏场景里的位置属性，以及显示效果如图 1-32 所示。如果修改位置属性里 Scale 在 X 属性上的值为原来的负数，游戏场景里的精灵就会“转身”，如图 1-33 所示。

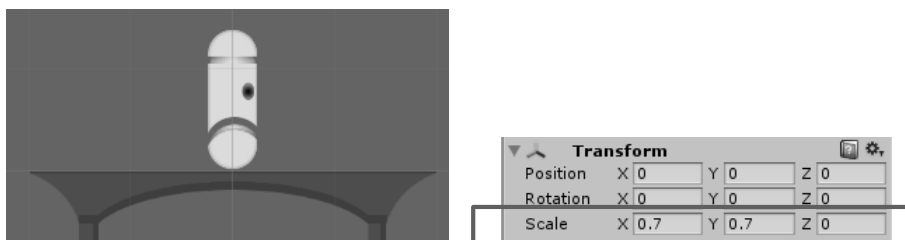


图 1-32 Scene 和 Inspector 视图，精灵面朝右

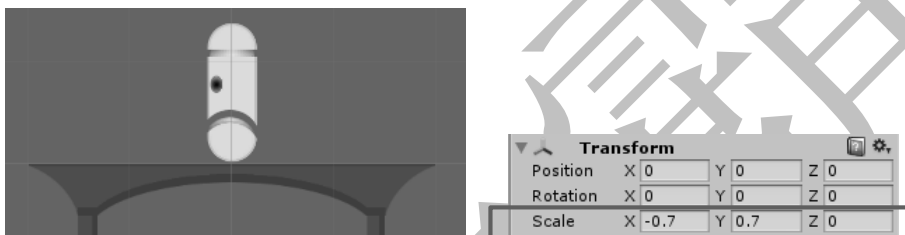


图 1-33 Scene 和 Inspector 视图，精灵面朝左

1.5 2D 游戏的运行效果

本章前前后后使用了很多节的篇幅，到底实现了怎样的一个游戏运行效果呢？或者说，游戏中的精灵会不会如我们所想的那样运行呢？关于这些疑问，会在本节集中揭晓。

(1) 单击 Unity 上方，工具栏里的  按钮，开始运行当前的游戏，默认精灵当前进入的是 Idle 动画状态，如图 1-34 所示。

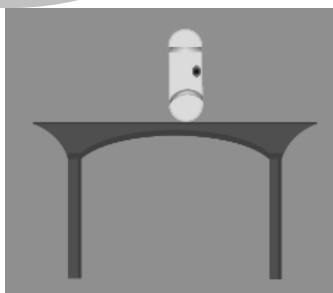


图 1-34 Idle 状态

(2) 当读者按下键盘上的左、右方向键，或者 A、D 键的时候，精灵会进入 Walking 动画状态，并且会向左或者向右移动，如图 1-35 所示。

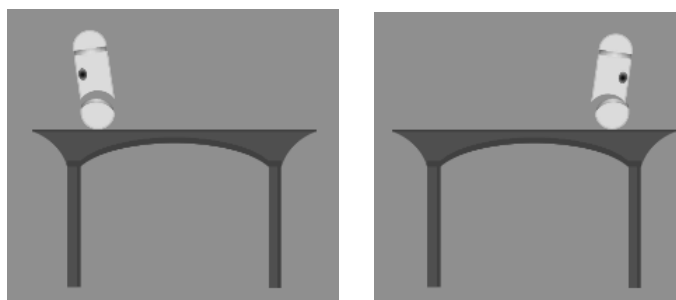


图 1-35 Walking 状态，左移和右移

(3) 当精灵移动到“地面”之外的時候，会发生“下落”，如图 1-36 所示。

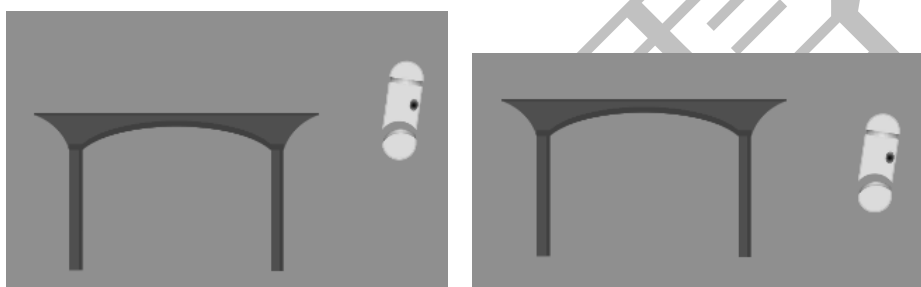


图 1-36 移动到“地面”之外，发生“下落”

提示：如果觉得精灵的移动速度太慢，可以修改 Player 对象上 Player State Listener 组件里 Player Walk Speed 属性的值，如图 1-37 所示，默认的值是 3。这个值越大，精灵的移动速度越快。如果觉得精灵的动画播放太慢，可以在 Animation 视图里设置指定动画的采样率，如图 1-38 所示。它的值越大，动画的播放速度越快。这两点在本章的前面有过说明，但是在这里决定还是要提醒下读者。



图 1-37 修改 Player 对象上，Player State Listener 脚本组件里的 Speed 属性，可以改变精灵的移动速度

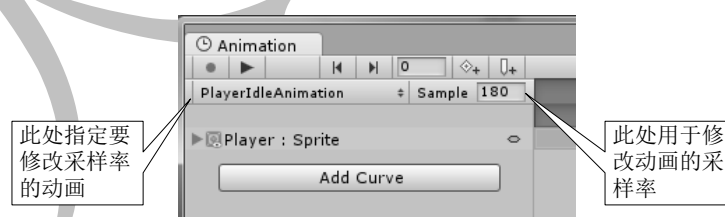


图 1-38 通过在 Animation 视图里，修改动画的采样率，进而修改动画的播放速度