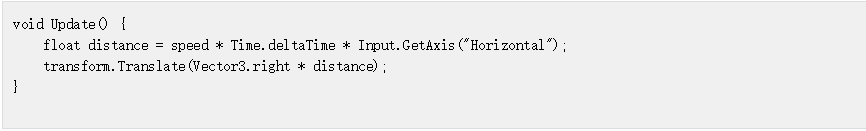
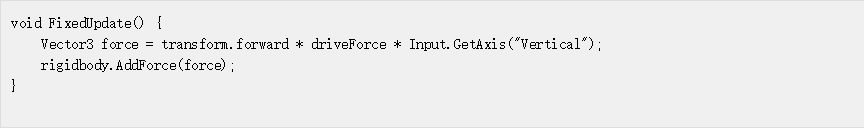
**事件函数**

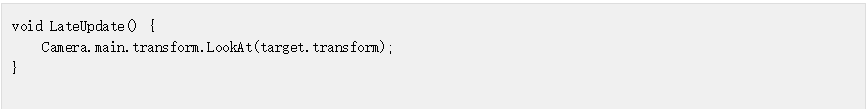
Unity 中的脚本与传统的程序概念不同。

在传统程序中，代码在循环中连续运行，直到完成任务。

相反，Unity 通过调用在脚本中声明的某些函数来间歇地将控制权交给脚本。函数执行完毕后，控制权将交回 Unity。这些函数由 Unity 激活以响应游戏中发生的事件，因此称为事件函数。Unity 使用一种命名方案来标识要对特定事件调用的函数。例如，先前介绍过的 Update 函数（在帧更新发生之前调用）和 Start 函数（在对象的第一次帧更新之前立即调用）。Unity 中提供了大量其他事件函数；可在 [MonoBehaviour](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.html) 类的脚本参考页面中找到事件函数的完整列表以及详细的事件函数用法说明。以下是一些最常见和最重要的事件。

游戏很像动画，其中的动画帧是动态生成的。游戏编程中的一个关键概念是在渲染每帧之前改变游戏对象的位置、状态和行为。[Update](file:///E:\UnityDocumentation\ScriptReference\MonoBehaviour.Update.html) 函数是 Unity 中包含这种代码的主要位置。在渲染帧之前以及计算动画之前都会调用 Update 函数。

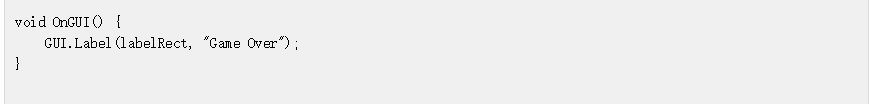
物理引擎也采用与帧渲染类似的方式以离散时间步骤进行更新。在每次物理更新之前都会调用一个称为 [FixedUpdate](file:///E:\UnityDocumentation\ScriptReference\MonoBehaviour.FixedUpdate.html) 的单独事件函数。由于物理更新和帧更新不会以相同频率进行，所以如果将物理代码放在 FixedUpdate 函数而不是 Update 中，此代码将产生更准确的结果。

为场景中的所有对象调用 Update 和 FixedUpdate 函数之后以及计算所有动画之后，如果能进行其他更改，也会很有用。一个例子是摄像机应该聚焦于目标对象；必须在目标对象移动后才能调整摄像机的方向。另一个例子是脚本代码应该覆盖动画的效果（例如，让角色的头部看向场景中的目标对象）。可使用 [LateUpdate](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.LateUpdate.html) 函数来处理这些类型的情况。

## 初始化事件

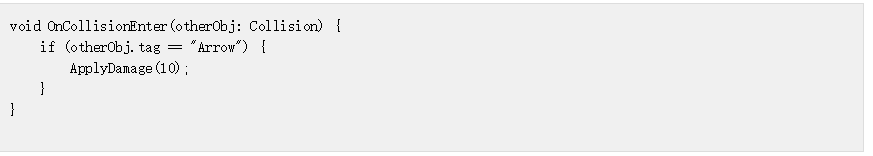
如果能在游戏运行过程中进行任何更新之前调用初始化代码，通常会很有用。在第一帧之前或开始对象的物理更新之前需要调用 [Start](file:///E:\UnityDocumentation\ScriptReference\MonoBehaviour.Start.html) 函数。场景加载时会为场景中的每个对象调用 [Awake](file:///E:\UnityDocumentation\ScriptReference\MonoBehaviour.Awake.html) 函数。请注意，虽然各种对象的 Start 和 Awake 函数的调用顺序是任意的，但在调用第一个 Start 之前，所有 Awake 都要完成。这意味着 Start 函数中的代码可以利用先前在 Awake 阶段执行的其他初始化。

## GUI 事件

Unity 有一个系统用于渲染场景中主要操作的 GUI 控件，并响应对这些控件的点击。此代码的处理方式与正常的帧更新有些不同，因此应将此代码置于定期调用的 [OnGUI](file:///E:\UnityDocumentation\ScriptReference\MonoBehaviour.OnGUI.html) 函数中。

此外，还可以检测场景中出现的游戏对象上发生的鼠标事件。使用此功能可以定位武器或显示当前在鼠标指针下的角色的相关信息。借助一系列 OnMouseXXX 事件函数（例如 [OnMouseOver](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnMouseOver.html)、[OnMouseDown](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnMouseDown.html)）可以让脚本对用户的鼠标操作做出反应。例如，如果指针位于特定对象上时按下鼠标按钮，则会调用该对象的脚本（如果存在）中的 OnMouseDown 函数。

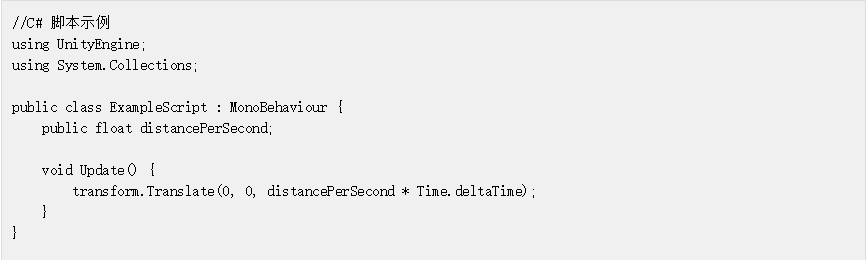
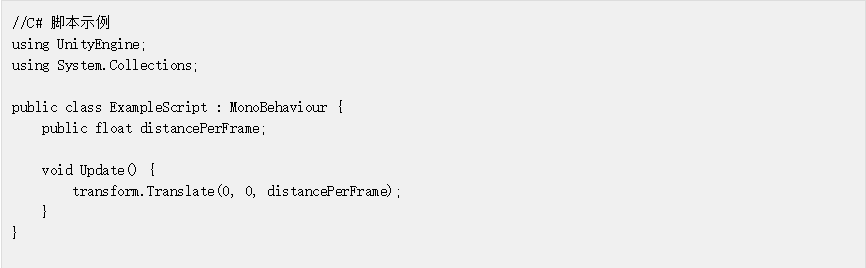
## 物理事件

物理引擎将通过调用该对象的脚本上的事件函数来报告对象的碰撞情况。在进行接触、保持接触和中断接触时，将调用 [OnCollisionEnter](file:///E:\UnityDocumentation\ScriptReference\MonoBehaviour.OnCollisionEnter.html)、[OnCollisionStay](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnCollisionStay.html) 和 [OnCollisionExit](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnCollisionExit.html) 函数。对象的碰撞体配置为触发器（即，碰撞体只检测某物何时进入而不进行物理反应）时，将调用相应的 [OnTriggerEnter](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnTriggerEnter.html)、[OnTriggerStay](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnTriggerStay.html) 和 [OnTriggerExit](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnTriggerExit.html) 函数。如果在物理更新期间检测到多次接触，可能连续多次调用这些函数，因此会将一个参数传递给函数，从而提供碰撞的详细信息（位置、进入对象标识等）。

# 时间和帧率管理

借助 [Update](file:///E:\UnityDocumentation\ScriptReference\MonoBehaviour.Update.html) 函数，可定期通过脚本监控输入和其他事件，并采取适当的操作。例如，可在按下“forward”键时移动一个角色。在处理这种基于时间的动作时要记住的一项重要规则是，游戏的帧率不是恒定的，并且 Update 函数调用之间的时间长度也不是恒定的。

举例来说，假设在一项任务中需要逐步向前移动某个对象，一次一帧。起初看起来好像可以在每帧将对象移动一个固定距离：

但是，如果帧时间不是恒定的，那么对象看起来会以不规则的速度移动。如果帧时间为 10 毫秒，那么对象将以 distancePerFrame 的距离每秒前进一百次。但如果帧时间增加到 25 毫秒（比如由于 CPU 负载的原因），那么对象每秒只会前进四十次，因此移动的总距离更短。解决方案是通过可从 [Time.deltaTime](file:///E:\\UnityDocumentation\\ScriptReference\\Time-deltaTime.html) 属性读取的帧时间来缩放移动距离大小：

请注意，此移动距离现在为 distancePerSecond 而不是 distancePerFrame。随着帧率的变化，移动步长大小也会相应改变，因此对象的速度将保持不变。

## 固定时间步长

与主帧更新不同，Unity 的物理系统\_会\_工作到固定的时间步长，这对于模拟的准确性和一致性很重要。在物理更新开始时，Unity 通过将固定的时间步长值添加到上次物理更新结束的时间来设置“警报”时间。然后，物理系统将执行计算，直到警报响起。

可从 [Time Manager](file:///E:\UnityDocumentation\Manual\class-TimeManager.html) 中更改固定时间步长的大小，并可使用 [Time.fixedDeltaTime](file:///E:\\UnityDocumentation\\ScriptReference\\Time-fixedDeltaTime.html) 属性从脚本中读取该值。请注意，较低的时间步长值将产生更频繁的物理更新和更精确的模拟，但代价是更大的 CPU 负载。除非要对物理引擎提出很高的要求，否则可能不需要更改默认的固定时间步长。

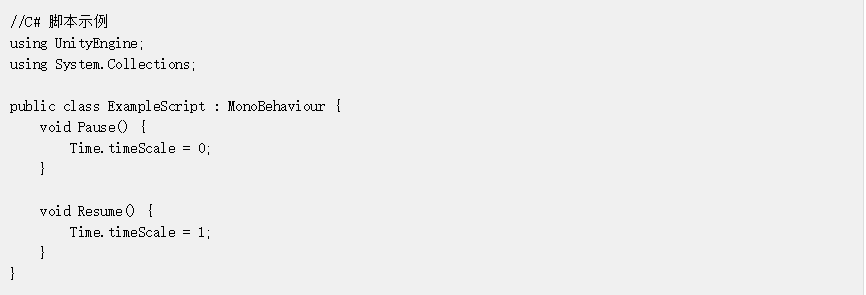
## Maximum Allowed Timestep

固定的时间步长使物理模拟能够实时保持准确，但是在游戏大量使用物理系统并且游戏帧率也变低的情况下（例如，由于游戏中存在大量对象），可能会导致问题。必须在常规物理更新之间“挤压”主帧更新处理；如果要进行大量处理，则可在单个帧期间进行多个物理更新。由于帧时间、对象位置和其他属性在帧开始时被冻结，因此图形可能与更频繁更新的物理系统不同步。

当然，只有这么多的 CPU 处理能力，但 Unity 可以选择让您有效地减慢物理时间，让帧处理能够赶上来。Maximum Allowed Timestep 设置（位于 [Time Manager](file:///E:\UnityDocumentation\Manual\class-TimeManager.html) 中）可限制 Unity 在指定的帧更新期间处理物理系统和 FixedUpdate 调用所花费的时间。如果帧更新花费的时间超过 Maximum Allowed Timestep 设置，则物理引擎将“让时间停止”并让帧处理赶上。一旦帧更新完成，物理引擎将恢复，就好像让时间停止后没有流逝一样。这种情况下的结果是刚体不会像正常情况下那样实时完美移动，而会稍微减慢。然而，物理“时钟”仍将跟踪它们，就像它们正常移动一样。物理时间的减慢通常是不明显的，并且是针对游戏性能的合理折衷。

## 时间标度

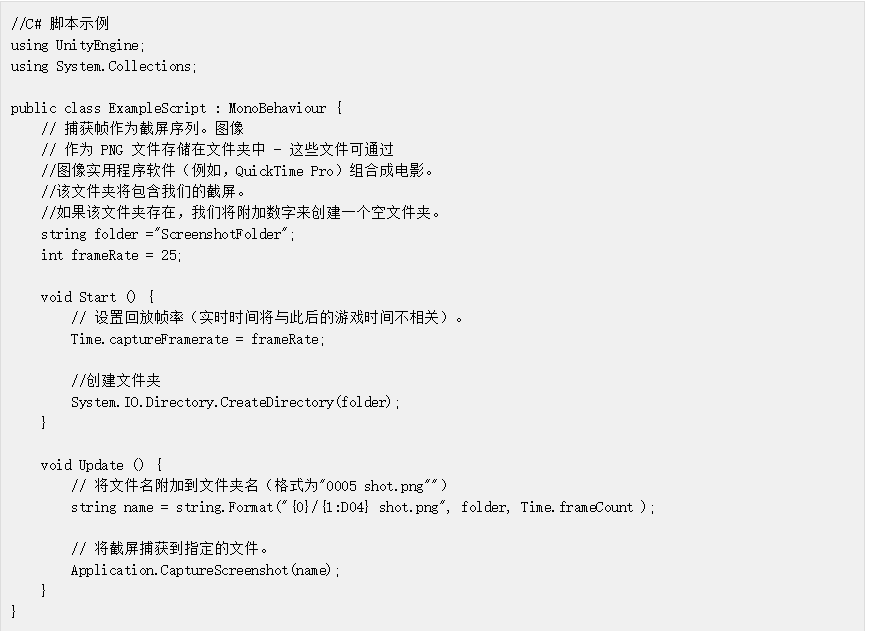
对于特殊效果，例如“子弹时间”，有时减慢游戏时间的流逝会很有用，能够使动画和脚本响应以较低的速率发生。此外，有时可能希望完全冻结游戏时间，就像游戏暂停时一样。Unity 有一个 Time Scale 属性可以控制游戏时间相对于实时时间的进展速度。如果该标度设置为 1.0，则游戏时间与实时时间匹配。值为 2.0 会使 Unity 中的时间流逝速度加倍（即，动作将加速），而值为 0.5 则会将游戏速度减半。值为零将使时间完全“停止”。请注意，时间标度实际上并不会降低执行速度，而只是更改了通过 [Time.deltaTime](file:///E:\\UnityDocumentation\\ScriptReference\\Time-deltaTime.html) 和 [Time.fixedDeltaTime](file:///E:\\UnityDocumentation\\ScriptReference\\Time-fixedDeltaTime.html) 报告给 Update 和 FixedUpdate 函数的时间步长。当游戏时间减慢时，调用 Update 函数的频率可能高于平常，但每帧报告的 deltaTime 步长将会缩短。其他脚本函数不受时间标度的影响，因此您可以在游戏暂停时显示具有正常交互的 GUI。

[Time Manager](file:///E:\UnityDocumentation\Manual\class-TimeManager.html) 有一个属性可用于全局设置时间标度，但使用 [Time.timeScale](file:///E:\\UnityDocumentation\\ScriptReference\\Time-timeScale.html) 属性从脚本设置该值通常更有用：

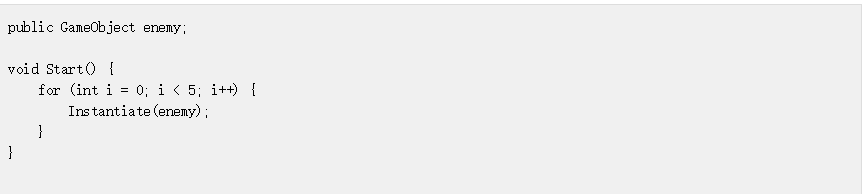
## Capture Framerate

一个非常特殊的时间管理案例是您希望将游戏过程录制为视频。由于保存屏幕图像的任务需要相当长的时间，因此如果在正常游戏过程中尝试执行此操作，则游戏的常规帧率将大幅降低。这将导致视频无法反映游戏的真实性能。

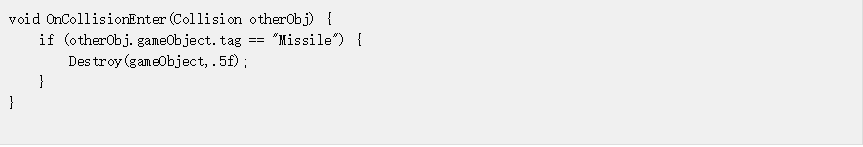
幸运的是，Unity 提供了一个 [Capture Framerate](file:///E:\UnityDocumentation\ScriptReference\Time-captureFramerate.html) 属性可用于解决该问题。该属性的值设置为零以外的任何值时，游戏时间将减慢，而帧更新将以精确的定期时间间隔发出。帧之间的时间间隔等于 1 / Time.captureFramerate，因此如果该值设置为 5.0，则每五分之一秒更新一次。随着对帧率的要求有效降低，在 Update 函数中便有了时间保存截屏或采取其他操作：



# 创建和销毁游戏对象

有些游戏在场景中保留恒定数量的对象，但是在游戏运行过程中创建和删除角色、宝藏和其他对象是很常见的做法。在 Unity 中，可以使用 [Instantiate](file:///E:\UnityDocumentation\ScriptReference\Object.Instantiate.html) 函数来创建游戏对象。该函数可以生成现有对象的新副本：

请注意，进行复制的源对象不必存在于场景中。更常见的做法是在 Editor 中使用从 Project 面板拖动到公共变量的预制件。此外，实例化游戏对象将复制原始对象中存在的所有组件。

此外，还有一个 [Destroy](file:///E:\UnityDocumentation\ScriptReference\Object.Destroy.html) 函数。该函数将在帧更新完成后或选择在短时间延迟后销毁对象：

