管理器

本页介绍了 TopDown Engine 中包含的各种管理器以及如何根据您的需要调整它们。

介绍

在**自上而下的引擎**使用管理为**中心参考点**进行了大量的类和组件。这些管理器**始终出现在您的场景中**，会记住当前的点数、播放的声音或生成角色的位置。

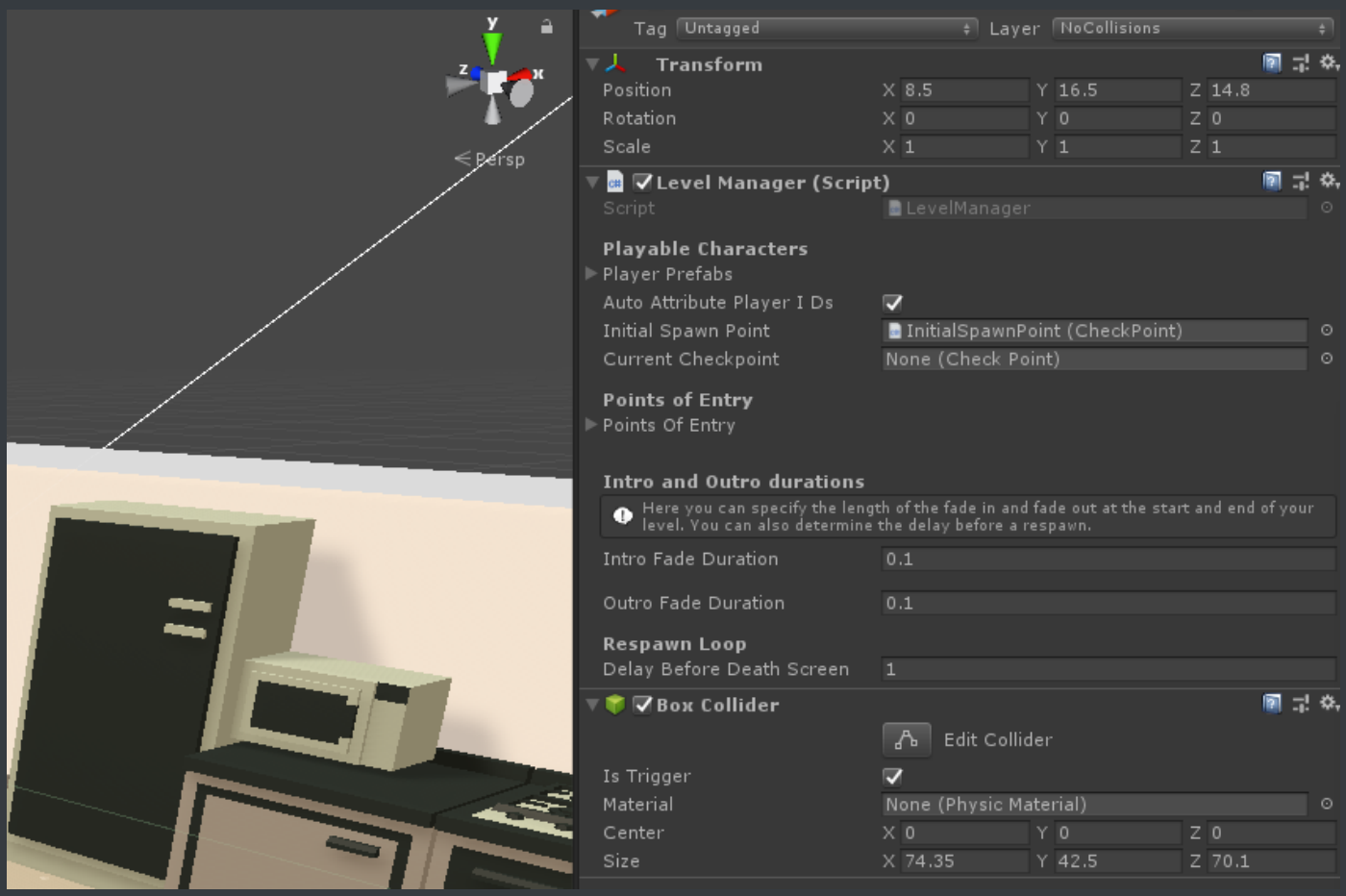
其中有一些，在大多数场景中，您将始终有游戏、关卡、输入、时间和声音管理器。通常你会想把它们放在空的游戏对象上。它们在场景中的位置无关紧要，反正它们是不可见的。将它们放在您的关卡之外是一种很好的做法，这样您就不会意外删除它们。

游戏经理

该**游戏管理器**是一个高层次的经理，负责制定目标帧速率，游戏循环，存储点或处理的时间表。它还可以选择性地处理生命系统，定义玩家可以拥有多少生命，以及它现在拥有多少生命。在这种情况下，您必须指定一个游戏结束屏幕以将玩家重定向到万一所有生命都丢失了。在大多数情况下，**您不必与它互动**，但请确保您的关卡中有一个。如果您想了解更多信息，API 文档是一个不错的去处。

级别经理

该**层管理器**是很多发动机的脚本的参考点。它维护游戏中的玩家角色列表（通常是一个，但如果您愿意，您可以拥有更多角色）。它还负责生成和重生以及检查点管理。



来自 Loft 演示场景的 LevelManager 的检查器

在大多数情况下，您需要在每个**关卡**中都有一个**LevelManager**。它需要在自己的游戏对象上。从它的检查器中，您可以（并且应该）设置要用作播放器的预制件。为此，展开 PlayerPrefabs 下拉菜单（如果它已折叠），只需将预制件从项目视图拖到 Element0 字段中即可。有关"自动属性玩家 ID"复选框的更多信息，请查看输入页面。

请注意，LevelManager 还允许您在场景中已有玩家角色。在这种情况下，只需清空 PlayerPrefabs 列表，然后将您的角色从场景中拖到 SceneCharacters 数组中即可。

您可以将 BoxCollider 添加到您的 LevelManager，然后它会自动充当您的相机的约束边界，只要它的虚拟相机上有一个 CinemachineCameraController。

从检查器中，您还可以指定延迟（在死亡屏幕出现之前，重生之前.....）以及用于淡入和淡出整个屏幕的淡入淡出设置。

输入管理器

该**输入管理**处理输入和发送命令给玩家。它在输入页面上有更详细的描述。

这个脚本的执行顺序必须是 -100，以确保它总是在每个 Update() 上首先运行，并避免奇怪的副作用，尤其是在低帧率下。如果您已将引擎导入到一个空白项目中，则应该是这种情况。您可以通过**单击脚本文件**，然后单击脚本检查器右下角的"**执行顺序"按钮**来定义脚本的执行顺序。有关更多详细信息，请参阅此页面。

声音管理器

引擎使用**MMSoundManager**系统来播放声音，无论是音乐、音效还是 UI 声音。

它的主要特点是：

- 播放/停止/暂停/恢复/自由声音
- 完全控制播放：循环、音量、音高、声像、空间混合、旁路、优先级、混响、多普勒水平、传播、滚降模式、距离
- 2D 和 3D 空间支持
- 内置池化，自动回收一组音频源以获得最佳性能
- 内置混音器和组，带有现成的音轨（主、音乐、SFX、UI），以及在需要时在更多组上播放的选项
- 停止/暂停/恢复/释放整个曲目
- 立即停止/暂停/恢复/释放所有声音
- 静音/设置音量整个轨道
- 保存和加载设置，内置自动保存/自动加载机制
- 淡入/淡出声音
- 淡入/淡出轨道
- 独奏模式：播放一首或所有曲目静音的声音，然后自动取消静音
- 用于干净 API 调用的 PlayOptions 结构
- 让声音在场景加载和场景之间持续存在的选项
- 轨道的检查器控件（音量、静音、取消静音、播放、暂停、停止、恢复、空闲、声音数

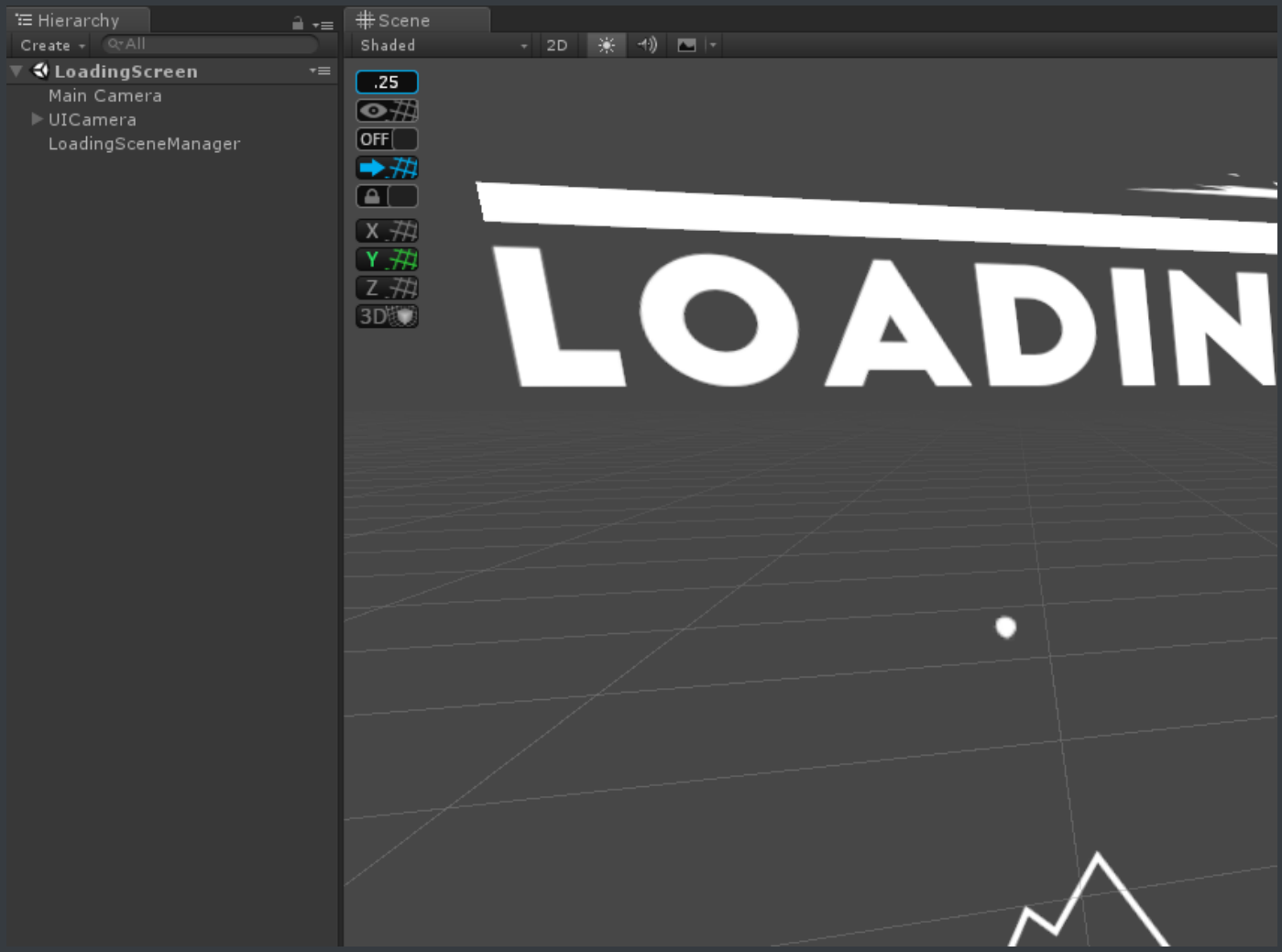
量）

- 与旧的 MM 系统和事件的复古兼容性，如 MMSfxEvents
- MMSoundManagerEvents：静音轨道、控制轨道、保存、加载、重置、停止持续声音

您可以在其专用文档中了解有关 MMSoundManager 的更多信息。

加载场景管理器

使用 Unity，通常当您想要转到另一个场景（例如在菜单中，或者从一个级别转到下一个级别）时，您会使用**SceneManager API**，并且可能使用 SceneManager.LoadScene() 方法。就我个人而言，我认为这种方法不会为玩家提供视觉反馈，例如在移动设备上加载场景可能需要几秒钟的时间，因此只有黑屏**并不能很好看**。该引擎带有自己的场景更改 API，如果您不喜欢它，您可以完全自由地**不使用**它。



场景视图中的 LoadingScene

如果您想为您的播放器提供更好的体验，您可以使用**加载场景管理器**：

- 它可以从**任何地方**调用，您的场景中不必有 LoadingSceneManager
- 它处理加载（顾名思义），显示**动画**和**进度条**
- 它是完全**可定制的**，只需编辑 Common/LoadingScene 场景的内容。您可以轻松添加自己的徽标、更改进度条的外观、正在播放的动画等。
- 这是很**简单的**来使用

要使用 LoadingSceneManager API，当您想要更改级别时，只需调用 LoadingSceneManager.LoadScene (string sceneToLoad) 方法。您传递的字符串参数当然必须与您尝试加载的场景的名称相**匹配**。因此，例如，如果您要加载 Koala Dungeon 级别，则可以使用：

```
LoadingSceneManager.LoadScene ("KoalaDungeon");
```

引擎会处理剩下的事情 

保存和加载管理器

该引擎带有一个简单的类来处理基本的保存和加载需求。该**MMSaveLoadManager**是一个静态类，允许对象的保存和加载在一个特定的文件夹和文件。它将让您将任何对象保存和加载到文件中。例如，您可以在 Inventory 和 MMAchievementManager 类中找到它的实际示例。

要保存对象，只需调用：

```
MMSaveLoadManager.Save(TestObject, FileName+SaveFileExtension,
FolderName);
```

然后加载它：

```
TestObject =
(YourObjectClass)MMSaveLoadManager.Load(typeof(YourObjectClass),
FileName + SaveFileExtension, FolderName);
```

还有一些帮助方法可以删除保存、保存文件夹等。不要犹豫，看看这个类，它像往常一样有完整的注释。您还可以指定系统应使用的 IMMSaveLoadManagerMethod。默认情况下它是二进制的，但您也可以选择二进制加密、json 或 json 加密。您将在 MMSaveLoadTester 类中找到有关如何设置其中每一项的示例

扩展管理器

如果出于某种原因您想修改经理的行为，最好的方法是扩展它。这样，如果您更新资产，您就不会丢失更改。扩展管理器相当容易。您所要做的就是创建一个新类，该类继承自您要更改其行为的经理，如下所示：

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

namespace MoreMountains.TopDownEngine
{
  public class NewGameManager : GameManager
  {
    protected override void Start()
      {
      base.Start();
      MMDebug.DebugLogTime("new game manager");
      }

    public override void Pause()
    {
      base.Pause();
      MMDebug.DebugLogTime("new game manager pause");
```

```
    }

    public override void AddPoints(int pointsToAdd)
    {
      base.AddPoints(pointsToAdd);
      MMDebug.DebugLogTime("new game manager add points");
    }
  }
}
```

在这个例子中，创建了一个新的 GameManager，并覆盖了它的一些方法。他们没有做太多事情，他们只是做基类所做的事情，然后输出调试信息。当然，你可以让他们做更多的事情。然后您要做的就是将新管理器放置在场景中的对象上，然后移除基础管理器。

您还可以决定向管理器的扩展版本添加新功能。像这样，例如：

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

namespace MoreMountains.TopDownEngine
{
  public class NewGameManager : GameManager
  {
    public virtual void Hello()
    {
      MMDebug.DebugLogTime("hello!");
    }
  }
}
```

在这种情况下，如果您想使用单例模式从另一个类调用该 Hello() 方法，请记住进行适当的转换，如下所示：

```
(NewGameManager.Instance as NewGameManager).Hello();
```