开放项目 | 公约

有用的链接

Chop Chop Github repo | 开放项目论坛 | Chop Chop Roadmap

编码

我们使用.NET标准作为起始基础，以下是我们自己的修改。

措辞

- 使用**描述性的**、**准确的**名称，即使这样会使名称变长。重视可读性而非简洁性。
- 不要使用**缩略语**。
- 当**缩略语**是一种公认的标准时，就使用这些**缩略语**。例如：UI、IO
- 方法名称应该是**动词**或**动词短语**。
- 属性名称应该是**名词**、**名词短语**或**形容词**。
- 布尔属性应该是**肯定性的**短语。你可以在布尔属性前加上 "是"、"有"、"能"。例如：IsActive, CanJump。
- 如果**多个属性**与同一个项目有关，使用**项目名称**作为前缀，并添加属性类型或角色。例如。

```
Color _gameTitleColor;
String _gameTitleString;
TextMeshProUGUI _gameTitleText;
```

- 避免使用**数字**作为名字，如果它们不是一个固有的列表的一部分。例如：*animator1*, *animator2*。取而代之的是**解释**两个属性之间的区别--例如 *playerAnimator*, *enemyAnimator*

资本化

定义

camelCase：第一个字母为小写。以下单词的第一个字母是大写的。

PascalCase：每个单词的第一个字母都是大写的。如果一个词是有两个字母的缩写，两个字母都大写。如果一个词是一个有两个以上字母的缩写，只有第一个字母是大写的。

- 类、方法、枚举、命名空间、公共字段/属性都是用**PascalCase**。例如：`ClassName`，`GetValue`。
- 本地变量、方法参数使用**camelCase**。例如：`previousValue`，`mainUI`。
- 私有字段/属性是camelCase，但以下划线开始。例子。`_inputReader`。
- 常量是全大写的，它们使用下划线来分隔单词。例子。`重力的数量`。

编程

- 保持字段和方法是**私有的**，除非你*需要*它们是公共的。
- 如果你想在检查器中暴露字段，而又不想让其他类实际访问该变量，请使用属性`[SerializeField]`和`private`，而不是把它们变成public。
  注意：这样做，你可能会得到 "字段从未被分配到，将永远有其默认值 "的警告。用`=default`给字段分配默认值。
- 尽量避免使用Singletons。探索使用ScriptableObjects (1, 2)，以获得一个类似的、集中的类，可以从多个对象中访问。
- 在声明一个变量时不要使用`var`。总是明确地写出它的类型。
- 避免使用静态变量。如果你绝对需要它们，确保它们与*快速进入播放模式*兼容，详见这里。
- 不要在你的代码中使用硬编码的 "神奇数字"。例如：玩家是通过`xInput * 0.035f`移动的。为什么是这个数字？取而代之的是，将数字存储在一个有明确名称的字段中--也许还有一个关于你为什么选择那个特定数字的注释。
- 对于 "using "指令（例如：`using System;`），在提交代码前，应删除所有未使用的指令。
- 如果你想把你的代码包围在一个命名空间中，使用 "UOP1 "来表示这个项目特有的代码和程序集。
  例如：`命名空间UOP1.Dialogues.Helpers。`

格式化

- 每列使用**1个Tab**来缩进代码，而不是空格。

- 大括号：如果它们是空的，它们应该在同一行。如果不是，它们应该在自己的行上，并在同一列中对齐。例如：

```
public class EmptyBraces(){ };
public class NonEmptyBraces
{
      //...
}
```

- 彼此之间包含的逻辑单元需要缩进，以表明层次关系。例如：

```
public void FunctionName()
{
    如果(somethingHappened)
    {
        //...
    }
}
```

评论

- **重要的是。**不要多此一举。如果你认为任何人只要看一下代码就能明白它的作用，就不要添加注释。相反，给你的变量、类和方法命名，这样它们就能解释自己了
- 使用内联评论来为每行代码提供额外的背景。
- 在每个类上面写一个摘要，描述该类的目的。可以选择包括关于该类如何工作的细节，特别是如果它不是特别直观或可读。例如。

```
// <summary>
/// 该类管理保存数据
// </summary>
```

提示：IDE通常在输入3次"/"符号时自动生成一个摘要。关于摘要的更多信息，请查看微软的官方规范。

- 在一个方法之前写一个注释，解释它的作用，以备名称不言自明或你想添加重要的细节。你也可以使用内联摘要。例如。

```
// <summary> 这个函数做这个... </summary> public float
CalculateBoundingBox(){ }
```

- 使用以`//TODO:`开头的注释来表示需要以后再接的东西，这样你就不会忘记它。
  注意：这不是邀请你去推送坏掉的功能。
- 不要使用`#区域`分隔符，或像`//`这样的 "行分隔符 "注释。 `----------`.

场景/层次结构

组织机构

- 在根部使用空的GameObjects作为分隔符来分割视觉上不同的逻辑部分。例如：---摄像机---，---环境---。 ----------照明 --- ...
  对这些对象应用EditorOnly标签，这样它们就会从构建中被剥离。
- 在有用的时候使用空的GameObjects作为容器，但如果它们只包含1-2个对象，就不要使用。
- UI。
  - 尽可能使用相同的画布，只有在画布属性改变时才创建多个画布。
  - 每个屏幕创建一个面板（主菜单、设置、暂停...）。
  - 使用面板作为容器来分组构成UI元素的部分。例如：一个设置标签和它的选项。

    面板也可以作为需要被固定在一起的元素的助手。例如：一些能量/物品的用户界面在屏幕的右下部分。

命名

- 不要在GameObjects的名称中使用空格。
- 使用**PascalCase**。例如：MainCharacter, DoorTrigger
- 当使用PascalCase会产生混淆时，使用下划线来连接两个概念。例如：MainHall_ExitTrigger, BossMinion_AttackWaypoints。

- **预制板。**重命名实例，如果它有意义的话。例如：一个预制变体文件被称为Protagonist_Scene1Variant，但一旦你使用它，你可以把它改名为Protagonist。

项目文件

命名

- 与场景/层次结构的规则相同。
- 为对象命名，这样当它们在同一个文件夹中并且有关联时，就**会自然地组合在一起**。
    - 一般来说，你以该对象所属的事物开始命名。例如：PlayerAnimationController、PlayerIdle、PlayerRun，等等。
    - 然而，当它有意义时，你可以为对象命名，使类似的对象呆在一起，即使它们与不同的 "所有者 "有关，或者如果形容词将它们分组不同。例如：在一个装满道具资产的文件夹中，你可以使用TableRound和TableRectangular，这样它们就会呆在一起；而不是RectangularTable和RoundTable。
- 避免在名称中出现文件类型。例如：使用ShinyMetal而不是ShinyMetalMaterial。

文件夹

§ 在根层，把你的资产放在确定游戏的区域/系统/地点的文件夹中。在那里，你可以创建子文件夹来分隔不同类型的资产。

§ 场景总是在一个根文件夹上

§ 脚本放在一个名为 Scripts 的根文件夹中，然后按系统分开。你可以在其中创建子文件夹，以便更好地对它们进行分类，定义ScriptableObjects的脚本应该放在这里。

§ 脚本对象（ScriptableObjects）本身应该放在一个名为相应的根文件夹中。

§ 当你制作一个新的系统时，你可以把一个**例子场景**放在/Scenes/Examples。尽量在这里收集相关的测试资产（Prefabs、Timelines、ScriptableObjects），不要只把它们放在项目的其他部分。

例子。

艺术
§字符
§§ PigChef

§§§ 材料
§§§ 预制板
§§§ 纹理
§环境
§§§ 室内装修
§§ 自然界
§§§ 材料
§§§预制板
§§§ 纹理
§§ 道具
场景
§ 示例
§地点
§ 菜单
ScriptableObjects（*数据*）
§音频
§ 存货项目
脚本
§音频
§活动
§ 库存系统
§§ 可脚本对象（*定义*）
§场景管理制度
UI
§ 材料