**Unity客户端开发规范**

**资源命名**

1. 包括图片、音乐、场景以及其它一切可能需要从服务上下载的资源，文件路径要求全部使用小写字母（linux服务器文件名区分大小写）。
2. 写代码时命名请使用英文，拼音太土了，对吧？注释可以使用中文，但切记在注释尾部加上英文空格，否则有极大概率引起unity3d编译错误；
3. 文件编码统一使用utf8，文件换行符统一使用\n换行；
4. 除非要兼容老代码或拥有非常明确的理由，请遵循命名规则。

**代码命名**

**摒除匈牙利命名法**

匈牙利命名法的命名规则带有成员变量的类型信息，这对需要经常重构的代码而言很可能会造成困惑，因为你可能修改了变量的类型但忘记了同步修改变量名：

// 以下为匈牙利命名法，摒除，不要使用

string strText;

int nAge;

float fSpeed;

bool bPlaying;

// 采用以下方式

string text;

int age;

float speed;

bool isPlaying;

**类命名规则**

1. 非public的类成员（包括变量与方法）统一以下划线开头，以便与方法参数区分；
2. 属性成员命名使用大写字母组合；
3. 无论是方法、属性还是成员变量，强制必须加private或public修饰符，不允许使用默认的修饰符；

class Game : MonoBehaviour

{

public static Game Instance // 属性成员由大写字母组合

{

get;

private set;

}

public int version = 0; // 当前需要在Unity中操作的MonoBehaviour子类的成员变量必须是public的，不需要带下划线

private GameSession \_gameSession; // private或protected的类成员变量命名以下划线开始，目的是区分成员变量与一般方法参数

}

统一将成员变量放到方法的后面，因为类的使用者通常更关心类有哪个方法可以供使用，而不会在意类的实现细节：

class Player

{

public void Tick(float deltaTime)

{

}

private int \_id;

private string \_name;

}

**枚举命名规则**

C#中使用枚举成员时必须通过“枚举名+点号+枚举成员”的形式，如JudgeType type = JudgeType?.Perfect;，不再采用C++中以全大写字母组合的方式。

public enum JudgeType

{

Perfect,

Cool,

Good,

Bad,

Miss,

Count

}

**代码风格**

**格式化**

代码格式化使用Microsoft Visual Studio风格，如果你使用MonoDevelop，请按以下方式设置：Preferences --> Source Code --> Code Formatting --> C# source code --> Policy下拉框，选择Microsoft Visual Studio

**括号使用**

// 即使只有一行代码，也要把双括号加上

if (true)

{

// only one line code;

}

// 禁止使用下面这种方式，因为在新添加if内部的逻辑时，很有可能会忘记补上忘记的括号

if (true)

// only one line code;

**foreach与循环**

**什么情况下foreach会产生gcalloc**

为了描述方便，我们将foreach遍历的对象称为"容器"。foreach并不等同于循环，它至少做了两件事情：

1. 通过调用容器的GetEnumerator()方法遍历容器；
2. 检查GetEnumerator()返回的迭代器是否实现了IDisposable接口，如果是，则通过该接口调用Dispose()方法；

而calloc就出现在第二步，将容器转换为IDisposable接口对象时。因此，关于如何回避foreach产生gcalloc的策略就是：

**策略一：禁用foreach**

只使用for循环和while循环实现容器遍历，而完全不使用foreach。 比如Array, List 可以使用for遍历，而其它的容器像Dictionary则可以使用collection.GetEnumerator?()遍历。

**策略二：了解foreach产生gcalloc的原因并按需使用**

1. 数组，这是一种非常特殊的容器，编译器会做特殊优化，在数组上使用foreach无gcalloc；
2. Unique.SortedTable?，是自己写的容器，做了特殊处理的，因此使用foreach无gcalloc；
3. 其它容器，禁止使用foreach；

**内存控制**

1. 当进行字符串进行格式化、连接等操作时，优先使用StringBuilder, string.Format(), string.Concat()等，不要直接进行字符串相加，以减少临时对象;
2. 由于struct的内存在栈上分配，不会引发GC，因此在方法内部使用而不需要长住内存的对象，优先考虑使用struct。使用struct传递参数时可以考虑使用ref，以回避copy代价；
3. 不要使用return new List<>()这种东西，考虑使用ref传参，或者将该List<>作为一个类的成员变量，并通过Clear()来维护数据；
4. 当心lambda表示式使用，不要让它变成一个closure。lambda表达式通常会编译成一个private方法，但如果lambda捕获了一个局部变量， 那它就不再是一个private方法， 它变成了一个closure， 此时，在每次调用lambda表达式的时候都会生成一个delegate对象；
5. 使用Dictionary时：
   1. 禁止使用foreach遍历，理由不再重述；
   2. 不要使用Dictionary<int, Cache<T>>这种泛型内套泛型的变量声明方式，IOS上AOT会出问题;
   3. 去学习一下ExtendedIDictionary扩展类，里面有许多方法使用会更加高效和方便；
   4. 遍历时使用var e = dict.GetEnumerator?()，使用pair.Key或pair.Value;
   5. 禁止定义为IDictionary<>这种接口的方式;
   6. 禁止使用dict.Keys或dict.Values;

**禁止使用的命名空间**

以下命名空间可以在编辑器代码里使用，但在游戏代码中禁止使用

1. UnityEditor? 没啥可说的，编译不过，无法导出；
2. System.Reflection 反射的东西，速度太慢，手机跑不动；
3. System.Xml 导出时会带一个1M大的dll，这个体积实在是太大了，可以考虑使用SmallXmlParser等小库处理；
4. System.Linq 迭代时会产生大量的临时对象，容易引发GC；

**禁止使用的方法**

1. StreamWriter?.WriteLine?(\*) 考虑使用Write('\n')代替，因为windows平台与mac平台产生的换行符不一样，需要保持跨平台性；