命名规则

- Camel Case:首字母小写，其余单词首字母大写
  - 应用：局部变量、函数参数
- Pascal Case: 全部单词首字母大写
  - 应用：类、函数、public fields
- Snake Case:用 _ 连接
- Kebab Case:用 - 连接
- Hungarian Notation: 前缀是变量的类型（Unity中不常见）

Camel case

Also known as camel caps, camel case is the practice of writing phrases without spaces or punctuation, separating words with a single capitalized letter. The very first letter is lowercase. Local variables and method parameters are camel case.

For example:
```
examplePlayerController
maxHealthPoints
endOfFile
```

Pascal case

Pascal case is a variation of camel case, where the initial letter is capitalized. Use this for class and method names in Unity development. Public fields can be pascal case as well. For example:
```
ExamplePlayerController
MaxHealthPoints
EndOfFile
```

Snake case

In this case, spaces between words are replaced with an underscore character. For example:
```
example_player_controller
max_health_points
end_of_file
```

Kebab case

Here, spaces between words are replaced with dashes. The words appear on a "skewer" of dash characters. For example:
```
example-player-controller
Max-health-points
end-of-file
naming-conventions-methodology
```

Hungarian notation

The variable or function name often indicates its intention or type. For example:
```
int iCounter
string strPlayerName
```

Hungarian notation is an older convention and is not common in Unity development.

- Unity内部的命名规则
  - bool类型的变量加上一个动词前缀例如is\has, isDead、isWalking、hasDamageMultiplier
  - 用能充分说明用途的名字，不要缩写
  - 用名词给变量起名，除非是bool值
  - public fields: Pascal Case
  - private fields: Camel Case
  - access level需要统一指出或统一省略
  - 可以在 私有成员变量 前加前缀比如 _ 、 m_ 以与 局部变量 区分，示例 private float _currentHealth;
- 类型很明确的时候，可以使用 var 提高可读性

```
// EXAMPLE: good use of var
var powerUps = new List<PowerUps>();
var dictionary = new Dictionary<string, List<GameObject>>();

// AVOID: potential ambiguity
var powerUps = PowerUpManager.GetPowerUps();
```

- 枚举类型，用单数名词，Pascal Case
  - 例外: bitwise，标注了 `System.FlagsAttribute` 属性的枚举类型用复数名词

```
// EXAMPLE: enums use singular nouns
public enum WeaponType
{
    Knife,
    Gun,
    RocketLauncher,
    BFG
}
public enum FireMode
{
    None = 0,
    Single = 5,
    Burst = 7,
    Auto = 8,
}

// EXAMPLE: but a bitwise enum is plural
[Flags]
public enum AttackModes
{
    // Decimal // Binary
    None = 0, // 000000
    Melee = 1, // 000001
    Ranged = 2, // 000010
    Special = 4, // 000100
    MeleeAndSpecial = Melee | Special // 000101
}
```

- 类和接口
  - 类名: Pascal Case
  - 一个文件只能有一个继承 `Monobehaviour` 的类，且类名与文件名需要匹配
  - 对于接口，用前缀 `I` +形容词

```
// EXAMPLE: Class formatting
public class ExampleClass : MonoBehaviour
{
    public int PublicField;
    public static int MyStaticField;
    private int _packagePrivate;
    private int _myPrivate;
    private static int _myPrivate;
    protected int _myProtected;

    public void DoSomething()
    {

    }
}
```

```csharp
// EXAMPLE: Interfaces
public interface IKillable
{
    void Kill();
}
public interface IDamageable<T>
{
    void Damage(T damageTaken);
}
```

- 函数
  - 函数名: Pascal Case
  - 参数: Camel Case

```csharp
// EXAMPLE: Methods start with a verb
public void SetInitialPosition(float x, float y, float z)
{
 transform.position = new Vector3(x, y, z);
}

// EXAMPLE: Methods ask a question when they return bool
public bool IsNewPosition(Vector3 currentPosition)
{
 return (transform.position == newPosition);
}
```

- 事件
  - 观察者模式: Observer pattern - Wikipedia
  - Use the `System.Action` delegate for events
  - Name the event with a `verb` phrase
  - Prefix the event raising method (in the subject) with "On"
  - Prefix the event handling method (in the observer) with the subject's name and underscore (_)
  - Create `custom EventArgs` only as necessary

```csharp
// EXAMPLE: Events
// using System.Action delegate
public event Action OpeningDoor; // event before
public event Action DoorOpened; // event after
public event Action<int> PointsScored;
public event Action<CustomEventArgs> ThingHappened;
```

```csharp
// raises the Event if you have subscribers
public void OnDoorOpened()
{
    DoorOpened?.Invoke();
}
public void OnPointsScored(int points)
{
    PointsScored?.Invoke(points);
}
```

```csharp
// define an EventArgs if needed
// EXAMPLE: read-only, custom struct used to pass an ID and Color
public struct CustomEventArgs
{
    public int ObjectID { get; }
    public Color Color { get; }
    public CustomEventArgs(int objectId, Color color)
    {
        this.ObjectID = objectId;
        this.Color = color;
    }
}
```

- 命名空间
  - Use `pascal case` without special symbols or underscores.
  - Add a `using directive` at the top of the file to avoid repeated typing of the namespace prefix.

```csharp
namespace Enemy
{
    public class Controller1 : MonoBehaviour
    {
        ...
    }

    public class Controller2 : MonoBehaviour
    {
        ...
    }
}
```

格式

Properties

- Properties是特殊的方法 (accessor)

```csharp
// EXAMPLE: expression bodied properties
public class PlayerHealth
{
 // the private backing field
 private int maxHealth;
 // read-only, returns backing field
 public int MaxHealth ⇒ maxHealth;
 // equivalent to:
 // public int MaxHealth { get; private set; }
}
```

```csharp
// EXAMPLE: expression bodied properties
public class PlayerHealth
{
 // backing field
 private int _maxHealth;
 // explicitly implementing getter and setter
 public int MaxHealth
 {
    get ⇒ _maxHealth;
```

```
    set ⟹ _maxHealth = value;
 }
 // write-only (not using backing field)
 public int Health { private get; set; }
 // write-only, without an explicit setter
 public SetMaxHealth(int newMaxValue) ⟹ _maxHealth = newMaxValue;
}
```

Serialization

- 对私有变量使用 `[SerializeField]` 属性
- 用 `[Range(min, max)]` 属性限定值的范围
- 用 结构体 包裹序列化的参数，inspector面板会更简洁

```
[Serializable]
public struct PlayerStats
{
    public int MovementSpeed;
    public int HitPoints;
    public bool HasHealthPotion;
}
```

括号与缩进

- 两种风格
  - Allman or BSD style(Indentation style - Wikipedia)
  - K&R style(Indentation style - Wikipedia)

```
// EXAMPLE: Allman or BSD style puts opening brace on a new line.
void DisplayMouseCursor(bool showMouse)
{
    if (!showMouse)
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    else
    {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}

// EXAMPLE: K&R style puts opening brace on the previous line.
void DisplayMouseCursor(bool showMouse){
    if (!showMouse) {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    else {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}
```

- 不要省略大括号（例如：for语句后、嵌套for语句）
- switch语句缩进格式

```
// EXAMPLE: indent cases from the switch statement
```

```csharp
switch (someExpression)
{
    case 0:
        DoSomething();
        break;
    case 1:
        DoSomethingElse();
        break;
    case 2:
        int n = 1;
        DoAnotherThing(n);
        break;
}
```

EditorConfig

- EditorConfig settings - Visual Studio (Windows) | Microsoft Learn
- EditorConfig
- .NET code style rule options - .NET | Microsoft Learn

行间空格

- 函数参数间，逗号后面加一个空格
- 不要在括号和参数间加空格

```csharp
// EXAMPLE: no space after the parenthesis and function arguments
DropPowerUp(myPrefab, 0, 1);
//AVOID:
DropPowerUp( myPrefab, 0, 1 );
```

- 不要在函数和括号间加空格

```csharp
// EXAMPLE: omit spaces between a function name and parenthesis.
DoSomething()
// AVOID
DoSomething ()
```

- 不要在中括号和参数间加空格

```csharp
// EXAMPLE: omit spaces inside brackets
x = dataArray[index];
// AVOID
x = dataArray[ index ];
```

- 流控制语句的括号前加一个空格

```csharp
// EXAMPLE
while (x == y)
// AVOID
while(x==y)
```

- 比较操作符两端加空格

```csharp
// EXAMPLE
if (x == y)
// AVOID
if (x==y)
```

- 控制一行的长度不要太长，(80-120 characters)
- 不要列队齐（持保留意见）

```
// EXAMPLE: One space between type and name
public float Speed = 12f;
public float Gravity = -10f;
public float JumpHeight = 2f;
public Transform GroundCheck;
public float GroundDistance = 0.4f;
public LayerMask GroundMask;

// AVOID: column alignment
public float            Speed = 12f;
public float            Gravity = -10f;
public float            JumpHeight = 2f;
public Transform        GroundCheck;
public float            GroundDistance = 0.4f;
public LayerMask        GroundMask;
```

列间空格

- 类似的方法放在一起
- 在下列情况可以加**两行空行**来区分
  - 变量声明与方法
  - 类与接口
  - if-then-else语块

#region

- #region指令可以折叠代码，但不是很有必要

代码格式化

- 格式化设置: Visual Studio (Windows): Tools > Options> Text Editor > C# > Code Style Formatting
- 格式化: Edit > Advanced > Format Document (Ctrl + K, Ctrl + D)

类

类的编写顺序

— Fields

— Properties

— Events / Delegates

— Monobehaviour Methods (Awake, Start, OnEnable, OnDisable, OnDestroy, etc.)

— Public Methods

— Private Methods

Single-responsibility principle

- 一个模块只为一件事负责
- 实践
  - 将一个既处理逻辑、数据、输入、音频等的 Monobehavior 类拆分，借助 ScriptObject
  - Unity - Manual: ScriptableObject (unity3d.com)
  - How to architect code as your project scales | Avoiding technical debt | Unity

Refactor a Paddle class into single responsibilities
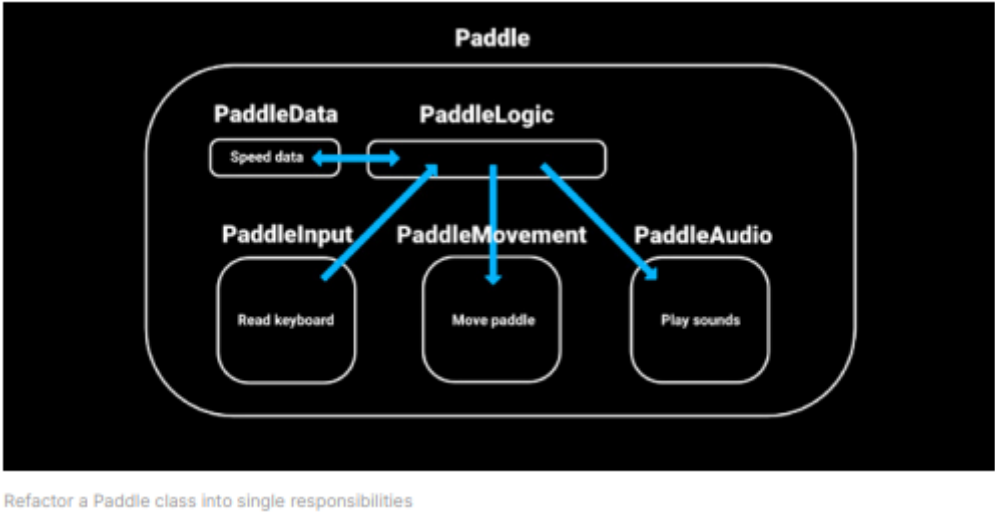
函数

设计原则

- 参数尽可能少
- 不要过度重载
- 不要根据传入的flag决定函数功能，将这种形式写成分开的两个函数
- 不要做多余的事

Extension methods

- 扩展方法 - Unity Learn

```
using UnityEngine;
using System.Collections;

//创建一个包含所有扩展方法的类
//是很常见的做法。此类必须是静态类。
public static class ExtensionMethods
{
    //扩展方法即使像普通方法一样使用，
    //也必须声明为静态。请注意，第一个
    //参数具有"this"关键字，后跟一个 Transform
    //变量。此变量表示扩展方法会成为
    //哪个类的一部分。
    public static void ResetTransformation(this Transform trans)
    {
        trans.position = Vector3.zero;
        trans.localRotation = Quaternion.identity;
        trans.localScale = new Vector3(1, 1, 1);
    }
}
```

```
using UnityEngine;
using System.Collections;

public class SomeClass : MonoBehaviour
{
    void Start () {
        //请注意，即使方法声明中
        //有一个参数，也不会将任何参数传递给
        //此扩展方法。调用此方法的
        //Transform 对象会自动作为
        //第一个参数传入。
        transform.ResetTransformation();
    }
}
```

```
}
```

注释

设计原则

- 避免过多的注释
- 换行写注释
- 在 `//` 和注释文本间加个空格
- 对于序列化的 `fields`，利用 Tooltip 属性代替注释

```
[Tooltip("The amount of side-to-side friction.")]
public float Grip
```

- xml 的注释

```
// You can also use a summary XML tag.
//
/// <summary>
/// Fire the weapon
/// </summary>
public void Fire()
{
    ...
}
```

ScriptTemplates

- 可以将 ScriptTemplate 目录复制进项目中，然后在项目中修改模板，修改后重启编辑器即可生效
- ScriptTemplate 中的文件名格式：`PriorityNumber-MenuPath-DefaultName.FileExtension.txt`
  - PriorityNumber：在目录中的顺序
  - MenuPath：可以用 `__` 来构造多级目录
- 例如：添加新的模板 `82-CustomScript__ScriptableObject-NewScriptableObject.cs.txt`，创建 ScriptableObject 脚本，通过该脚本可以再从目录中选择创建 ScriptableObject

```
using UnityEngine;

[CreateAssetMenu(fileName = "#SCRIPTNAME#", menuName = "ScriptableObjects/#SCRIPTNAME#", order = 1)]
public class #SCRIPTNAME# : ScriptableObject
{
#NOTRIM#
}
```

测试

- [About Unity Test Framework | Test Framework | 1.1.33 (unity3d.com)](#)
- [Workflow: How to create a new test assembly | Test Framework | 1.1.33 (unity3d.com)](#)
- [https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/workflow-run-playmode-test-standalone.html](#)

参考资料

- [Framework Design Guidelines | Microsoft Learn](#)
- [C# Coding Conventions | Microsoft Learn](#)
- [C# at Google Style Guide | styleguide](#)
- [Unity - Manual: Script serialization (unity3d.com)](#)
- [10 ways to speed up your programming workflows in Unity with Visual Studio 2019 | Unity Blog](#)

- [Framework Design Guidelines | Microsoft Learn](#)

- [C# Coding Conventions | Microsoft Learn](#)

- [C# at Google Style Guide | styleguide](#)

- [Unity - Manual: Script serialization (unity3d.com)](#)

- [10 ways to speed up your programming workflows in Unity with Visual Studio 2019 | Unity Blog](#)