Orange Games — Coding Standards

“The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you are saying, rather than on how you are saying it.” — Google C++ Style Guide

---

1. Purpose

This document defines the coding standards used by the **Orange Games** development team.

Our goals are: - Consistency across all scripts - Readable and maintainable code - Fewer merge conflicts - Easier collaboration and debugging

All Unity C# scripts (Player, Enemy, UI, Camera, etc.) should follow this style guide.

---

2. Naming Conventions

General Rules

- Use descriptive names that reflect purpose.
- Avoid single-letter names except for short-lived iterators (`i` , `j` , `k`).
- Use English words and avoid abbreviations unless well known (`UI` , `ID`).

Variables

- Use **camelCase** for variables.
- Prefix Boolean variables with `is` , `has` , or `can` .
- Use postfix like `_Count` or `_Speed` for measurements or counters.
- Use an underscore `_` prefix for private serialized fields visible in the Inspector.

Example:

```
[SerializeField] private float _moveSpeed;
int enemyCount;
float jumpForce;
bool isGrounded;
```

Constants

- Write constants in **ALL_CAPS** with underscores.

```
const float MAX_SPEED = 10f;
const int MAX_HEALTH = 100;
```

Classes, Structs, Enums

- Use **PascalCase**.

```
public class PlayerController { }
public struct PlayerStats { }
public enum PlayerState { Idle, Moving, Jumping }
```

Methods and Functions

- Use **PascalCase**.
- Function names start with a verb that describes the action.

```
void HandleInput() { }
void ApplyDamage(int amount) { }
bool IsAlive() { return health > 0; }
```

---

3. Commenting Style

File Header Comment

Every file must start with a header block:

```
/*****************************************
 * File: PlayerController.cs
 * Author: Orange Games Team
 * Description: Handles player input, jumping,
 * attacking, and damage for the Orange Games project.
 *****************************************/
```

Function Header (Updated)

Use **triple-slash comments** directly above the method:

```
/// Handles player movement and attack logic
```

Inline Comments

Use `//` for short explanations.

```
// Apply horizontal movement based on input
rb.linearVelocity = new Vector2(move * moveSpeed, rb.linearVelocity.y);
```

Only comment **why** something happens, not what obvious code does.

---

4. Indentation and Formatting

- 4 spaces per indentation level (no tabs).
- Braces `{}` on the same line as the condition or method.
- One blank line between methods.

Example:

```
void Update() {
    HandleInput();
    if (!playerData.IsAlive()) {
        Die();
    }
}
```

Conditional Formatting

```
if (condition) {
    // statements
} else if (otherCondition) {
    // statements
} else {
    // statements
}
```

Loop Formatting

```
for (int i = 0; i < enemyCount; i++) {
    // iterate through enemies
}
```

---

5. Error Handling

- Use `Debug.Assert()` for logic validation.
- Use `Debug.LogError()` or `Debug.LogWarning()` for runtime errors.
- Never leave an empty `catch` block.
- Prevent errors instead of catching them.
- Always check for `null` before using components.

Example:

```
void Start() {
    rb = GetComponent<Rigidbody2D>();
    Debug.Assert(rb != null, "Rigidbody2D missing on Player!");
}

void ApplyDamage(int amount) {
    Debug.Assert(amount >= 0, "Damage must be non-negative!");
}
```

---

6. Unity Lifecycle Usage

Method	Purpose
Awake()	Initialize references and singletons
Start()	Initialize gameplay variables
Update()	Handle per-frame logic
FixedUpdate()	Handle physics updates
OnDrawGizmosSelected()	Debug visuals (attack range, etc.)

Example:

```
void Awake() {
    rb = GetComponent<Rigidbody2D>();
}

void Start() {
    playerData = new Player(100);
}

void FixedUpdate() {
```

```
    HandleMovement();
}
```

---

7. File and Folder Organization

Project Folder Structure:

```
Assets/
 ├── Scripts/
 │    ├── Player/
 │    │    ├── Player.cs
 │    │    ├── PlayerController.cs
 │    │    └── FollowPlayer.cs
 │    ├── Enemy/
 │    │    └── EnemyController.cs
 │    ├── UI/
 │    │    └── HealthBar.cs
 │    ├── Levels/
 │    └── Audio/
 └── Tests/
      ├── TeamLead1/
      ├── TeamLead2/
      ├── TeamLead3/
      ├── TeamLead4/
      │    └── PlayerStressTest.cs
      └── TeamLead5/
```

Rules: - One public class per file. - File name must match the class name exactly. - Separate gameplay and testing code into different folders.

---

8. Debugging and Logging

- Use `Debug.Log()` for general messages.
- Use `Debug.LogWarning()` for potential issues.
- Use `Debug.LogError()` for critical problems.
- Remove or comment out logs before the final release.

Example:

```
Debug.Log("Player jump successful");
Debug.LogWarning("Player out of bounds!");
Debug.LogError("Missing Rigidbody2D component!");
```

---

9. Example of Clean Code

```
/*****************************************
 * File: ExampleManager.cs
 * Author: Orange Games Team
 * Description: Demonstrates standard code structure,
 * formatting, comments, and naming conventions.
 *****************************************/

using UnityEngine;

/// Example class showing clean, maintainable code structure
public class ExampleManager : MonoBehaviour
{
    [SerializeField] private float _speed = 5f;
    [SerializeField] private int _maxScore = 10;

    public int currentScore = 0;

    /// Initializes the component when the game starts
    void Start() {
        Debug.Log("ExampleManager initialized.");
    }

    /// Reads player input
    void HandleInput() {
        // Input handling
        if (Input.GetKeyDown(KeyCode.Space)) {
            AddScore(1);
        }
    }

    /// Adds score when space is pressed
    void AddScore(int amount) {
        Debug.Assert(amount > 0, "Score increment must be positive!");
        currentScore += amount;
        Debug.Log("Score increased to: " + currentScore);
    }
}
```

10. Summary Table

Category	Rule
Variable Names	camelCase (`moveSpeed`, `isGrounded`)
Constants	ALL_CAPS with underscores (`MAX_HEALTH`)
Classes & Methods	PascalCase (`PlayerController`, `HandleInput()`)
Indentation	4 spaces
Braces	Same line
Comments	File Header + Inline + Triple-Slash
Error Handling	`Debug.Assert`, `Debug.LogError`
Unity Rules	Awake → Start → Update → FixedUpdate

Orange Games © 2025 — Internal Development Guide