Unity: стандарт кодирования

Иванова М., Бурков Э.

Оглавление

[**Общие правила** 1](#_Toc479187427)

[**Оформление кода** 1](#_Toc479187428)

[**1.** **Фигурные скобки** 1](#_Toc479187429)

[**2.** **Пробелы** 2](#_Toc479187430)

[**3.** **Именование объектов** 3](#_Toc479187431)

[**4.** **Форматирование параметров функции** 4](#_Toc479187432)

[**5.** **Организация файлов** 4](#_Toc479187433)

[**6.** **Длина строки** 5](#_Toc479187434)

[**7.** **Инициализация новых объектов** 5](#_Toc479187435)

[**8.** **Комментарии** 5](#_Toc479187436)

[**Использование Unity** 5](#_Toc479187437)

[**Источники** 6](#_Toc479187438)

# **Общие правила**

1. Лучше писать код медленнее, то более качественно.
2. Лучше писать код медленнее, но сделать его более легким в поддержке
3. Старайтесь сократить объем кода там, где это возможно. Но важно, чтобы сокращения не приводили к усложнениям для понимания другими разработчиками.
4. При компиляции готового кода не должно быть warning’ов.

# **Оформление кода**

## **Фигурные скобки**

* 1. Фигурные скобки обязательно должны располагаться каждая *на отдельной строке*. Открывающая скобка всегда находится в начале строки, следующей за той, которая открывает блок. Закрывающая должна быть в конце блока визуально под открывающей.

**if** (someExpression)

{

DoSomething();

}

**else**

{

DoSomethingElse();

}

* 1. Все содержимое блока между фигурными скобками должно быть выделено *4 пробелами* (табы нельзя использовать)
  2. При использовании оператора *switch* нужно использовать фигурные скобки. Выражения в *case*, содержащие несколько строк, *должны быть также обрамлены фигурными скобками*.

**switch** (someExpression)

{

**case** 0:

DoSomething();

**break**;

**case** 1:

DoSomethingElse();

**break**;

**case** 2:

{

**int** n = 1;

DoAnotherThing(n);

}

**break**;

}

* 1. Фигурные скобки *должны быть* даже если *блок содержит одну строку кода*. Это позволяет сделать код более читабельным.
  2. *Тело «пустых» методов* должно занимать *две строки*

**void** EmptyMethod()

{

}

* 1. Возможно написание свойства (get/set конструкции) в 1 строку, если оно используется как автоматическое.

public int Var { get; set; }

## **Пробелы**

* 1. Одинарные пробелы должны использоваться *после запятой между аргументами функции или метода*

Console.**In**.Read(myChar, 0, 1);

* 1. Одинарный пробел используется *перед сторожевым условием на исполнение блока кода*

**while** (x == y)

* 1. Одинарный пробел *перед и после оператора сравнения*

**if** (x > y)

* 1. Одинарные пробелы используются для *разделения арифметических операций и операндов.*

**int** value = (number + 2) \* 10;

* 1. *Нельзя* использовать пробелы *после скобок или между аргументами функции и запятыми после них*

CreateFoo(myChar, 0, 1)

* 1. *Нельзя* использовать пробелы *между именем функции и скобками*

CreateFoo()

* 1. *Нельзя* использовать пробелы *внутри квадратных скобок*

x = dataArray[index];

* 1. *Нельзя* использовать пробел *между производными типами*

**var** list = **new** List<int> ();

## **Именование объектов**

* 1. Имена объектов должны быть *легко читаемы* и *соответствовать назначению объекта*. Нельзя использовать числа как имена параметров. Если непонятно, как назвать параметр *унарной функции*, то можно использовать имя *value*. Для параметров *бинарной функции* можно использовать *left и right*.
  2. *Нельзя* использовать *венгерскую нотацию* при именовании объектов.
  3. *Нельзя* использовать *префиксы перед именами переменных*, *перечислений и классов* (\_, m\_, s\_ и т.д.). Если требуется *отделить локальные переменные от параметров методов*, *используйте this*. Используйте this только в случае крайней необходимости.
  4. Для имен *локальных переменных и параметров методов* используйте *camelCase*.
  5. Для *имен методов, свойств, событий, типов, пространств имен, перечислений и имен классов* используйте *PascalCasing*.
  6. Используйте *префикс “I”* перед *именами интерфейсов*.
  7. Имена *классов и структур* должны быть *существительными*. В именах *интерфейсов* могут использоваться *прилагательные и существительные*. В именах *методов* используются только *глаголы и глагольные выражения.*
  8. *Нельзя* называть методы, начиная *со слова get*, если это не стандартные геттеры.
  9. Имена *булевских переменных* должны быть только *в утвердительной форме*. Можно использовать “is”, “can” или “has” в начале имени там, где это имеет смысл.
  10. Имена ресурсов должны заканчиваться на то существительное, которое говорит о цели использования данного ресурса. (MyRes1Texture, MyRes2Model, …).
  11. Глобальные переменные именуются с использованием *camelCase* и начинаются со слова *global*. (globalMyVar, globalValue).
  12. Имена констант должны быть в UpperCase формате. Если наименование многосложное используется ‘\_’ для разделения членов. (CONSTANT, MY\_CONST\_VAR).
  13. В наследовании/имплементации порядок наследуемых/имплементирующих классов должен быть алфавитным. (class A : IAInterface, IBInterface, ICInterface).

## **Форматирование параметров функции**

* 1. Если параметров функции слишком много и их требуется перенести на другую строку, то новая строка параметров должна начинаться там же, где начинаются параметры на строку выше.

CreateLine(format, foo,

bar, baz)

* 1. При переносе параметров на новую строку запятая должна оставаться в конце строки, а не в начале.

CreateLine(format,

foo,

bar,

baz)

## **Организация файлов**

* 1. Исходные файлы могут содержать только один публичный класс. В противном случае файл требуется разделить на несколько (по количеству публичных классов). В публичном классе могут быть несколько вложенных классов.
  2. Имя исходного файла должно совпадать с именем публичного класса, определенного в этом файле.
  3. Все члены класса должны быть отсортированы по алфавиту и разбиты на смысловые группы (поля, конструкторы, свойства, события, методы и т.д.)

**using** System;

**using** UnityEngine;

**public** **class** MyClass : MonoBehavior

{

*// fields*

**int** foo;

*// properties*

**public** **int** Foo { get { … } set { … } }

*// methods*

**void** MyMethod(**int** number)

{

**int** value = number + 2;

Debug.Log(value);

}

}

## **Длина строки**

* 1. Используйте строки кода длиной не более чем в 120 символов.

## **Инициализация новых объектов**

* 1. При создании новых объектов класса значение полей задаются либо с помощью уже созданного конструктора, либо отдельно определяя значение каждого поля. Нельзя определять новое тело конструктора при создании нового объекта.

## **Комментарии**

* 1. Комментарии должны описывать основную идею следующих за ним строк, суть алгоритма или логический поток. Писать комментарии надо так, чтобы любой сторонний пользователь должен, прочитав комментарии, понять работу основных функций и операций.
  2. Для *однострочных* комментариев используется *//*. По возможности комментарии должны *предшествовать коду*, к которому они относятся.
  3. Если комментарий *достаточно короткий* (содержит 2-3 слова), то он может быть записан *после выражения*, к которому он относится*, в той же строке*.
  4. Комментарии, *занимающие несколько строк*, должны быть оформлены следующим образом:

*/\**

*\* Blah*

*\* Blah again*

*\* and another Blah*

*\*/*

*//*

*// Blah*

*// Blah again*

*// and another Blah*

*//*

* 1. При описании метода можно использовать: *емкий комментарий*, описывающий его цель, или *подробный комментарий*, описывающий параметры метода и возвращаемое значение.

*/\* Func name.  
 \* > ‘Var var1’ – var for fun.  
 \* > ‘int num’ – num of cool.  
 \* < nothing/’List’ - cool list of things  
 \*/*

# **Использование Unity**

1. Лучше пере-использовать выделенные ресурсы, вместо того, чтобы создавать новые.

# **Источники**

<http://wiki.unity3d.com/index.php/Csharp_Coding_Guidelines>

<https://prezi.com/povbmlr5-w8j/coding-standards-for-game-development-in-unity3d/>

<http://www.mono-project.com/community/contributing/coding-guidelines/>

<https://msdn.microsoft.com/en-us/library/ms229002.aspx>