FIERCE FUN

Fiercefun.com

Coding Conventions: 1.03

Please note that these conventions are not optional. Any project not following these conventions will not be used in production.

Coding conventions are style guidelines for programming. They typically cover:

- Naming and declaration rules for variables and functions.
- Rules for the use of white space, indentation, and comments.
- Programming practices and principles

Fierce Fun primarily uses C# and scripting languages (ActionScript, JavaScript, PHP) so our coding conventions follow those used primarily in Java/JavaScript standards.

The Importance of Naming

Choosing good names for variables, functions and classes is critical to creating code that is easy to use and easy to understand. You should always take the time to think about whether you have chosen the right name for something.

Properly naming and commenting items is one of the most critical task of software projects. At all times, developers should care about and adhere to this critical task.

Version Control

GitHub

Unity Collaborate

Links

Our conventions are based on the following

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions C#

https://pear.php.net/manual/en/standards.php (PHP)

General Coding Practise

At Fierce Fun, we classify code as development and production.

<u>Development Code</u>
Development is what you work on day-to-day. It can be messy at time, undocumented and buggy. Effectively it is your in-progress work and only you will see it.

<u>Production Code</u>
Production code is what you submit for review on a weekly basis. This code could be used in the final production build so it should be clean, organised and commented before submitting. It should be designed to be easily read and understood by another developer. If it is not easily read and understood, it will not be usable as production code.

Coding Guidelines

Code should not be smelly! Examples of nasty smells are:

- Repeated code
- Big class size
- Big function size
- Deep nesting with if statements
- Long line length
- No or obscure comments

Please read the follow important guidelines to avoid smelly code
https://en.wikipedia.org/wiki/Code_smell

If it smells, we can’t use it….

Copyright Notice

Include this copyright notice at the top of every file

```
/**
 * File Name:   Playza.php
 * Class:       Playza
 *
 * Description: Root class of the administration system, used primarily for logging data, error reporting
 *                          NB, only use with base PCS classes, use Playza_Log for other classes
 *
 * @copyright  Fierce Fun Ltd
 * @version    1.0
 * @created    01/04/2014
 */
```

For code files, use

```
/**
 * Class:                     Avatar
 *
 * Description:              Class to initialise & update main game player. This class forms the basis for the player
*                            and enemy data structres
*
* Copyright                  Fierce Fun Ltd
* Version                    1.0
 */
```

Code without this notice cannot be used.

General Naming Conventions

<u>Class and Method Names</u>
use PascalCasing for class names and method names.

```
public class GameManager
{
    public void ClearScreen()
    {
        //...
    }
    public void DrawScreen()
    {
        //...
    }
}
```

<u>Variables and Arguments</u>
Use **camelCase** for identifier names. For example, select myPelican rather than mypelican.

<u>Examples</u>
firstName = "John";
lastName = "Doe";
fullPrice = 99;

Name a file by describing the process or item, such as add-user.

Remember to keep names descriptive.

<u>Abbreviations</u>
Avoid them as a general rule. For example, calculateOptimalValue() is a better method name than calcOptVal().

Code Layout

Brace Style

Vertically align curly brackets. Braces shall be used for all blocks in the style shown here:

```
{
	code here…
	code here...
}
```

Example:

package samples

```
{
	public class SampleCode
	{
		public var sampleGreeting:String;

		public function sampleFunction()
		{
			trace(sampleGreeting + " from sampleFunction()");
		}
	}
}
```

Member Variable Declaration

Declare all member variables at the top of a class, with static variables at the very top.

```
public class Account
{
    public static string PlayerName;
    public static decimal Reserves;

    public string Number {get; set;}
    public DateTime DateOpened {get; set;}
    public DateTime DateClosed {get; set;}
    public decimal Balance {get; set;}

    // Constructor
    public Account()
    {
        // ...
    }
}
```

Variables must be prefixed e.g. public, private or protected

Spacing

There should be a 3-line gap between all functions/methods

```
// Constructor
public function Main()
{
    // standard initialisation
    if (stage) init();
    else addEventListener(Event.ADDED_TO_STAGE, init);
}



// App initialisation - initial state
private function InitApp(e:Event = null):void
{
    // entry point
    removeEventListener(Event.ADDED_TO_STAGE, init);

    // go to Into screen (only called once)
    currentState = STATE_INTRO;
    intro = new Intro();
    intro.addEventListener( StartEvent.START, onBtnStart, false, 0, true );
}



// Menu screen: respond to 'start' button on Intro screen - go to menu screen
protected function OnBtnStart(startEvent:StartEvent ):void
{
    var inputtedText:String = startEvent.getTextValue();

    // add new menu screen object
    menu = new Menu(inputtedText);
    menu.addEventListener( NavigationEvent.VIEW_TEXT, onGoDisplay, false, 0, true );

    // dispose of intro screen object (for garbage collectin)
    intro.dispose();
    intro = null;
}



/**
```

Commenting

In general, code should be commented prolifically. It should allow an external developer to understand your code.

What needs to be commented:

- Class/Script Heading
- All member variables
- All functions/methods
- Important lines in a function

Class/Heading comments

This sentence/paragraph describes what the class/script does.

```
import flash.net.LocalConnection;
import flash.system.System;


/**
* Class:        Main
*
* Description:  Main is an instance of the Document Class 'DC' (the top-level class associated with the application)
*               A DC must inherit from either Sprite or MovieClip and represents the main application timeline
*
* Copyright     Fierce Fun Ltd
* Version    1.0
*/
public class Main extends MovieClip
{
```

Class Variable/Members

With variable declaration (as there be many), it is ok to comment at the end of the line

```
public float moveSpeed = 2f;          // The speed the enemy moves at.
public int HP = 2;                    // How many times the enemy can be hit before it dies.
public Sprite deadEnemy;              // A sprite of the enemy when it's dead.
public Sprite damagedEnemy;           // An optional sprite of the enemy when it's damaged.
public AudioClip[] deathClips;        // An array of audioclips that can play when the enemy dies.
public GameObject hundredPointsUI;    // A prefab of 100 that appears when the enemy dies.
public float deathSpinMin = -100f;            // A value to give the minimum amount of Torque when dying
public float deathSpinMax = 100f;             // A value to give the maximum amount of Torque when dying
```

Comments inside functions

- Place the comment on a separate line, not at the end of a line of code.
- Begin comment text with an uppercase letter.

Build Numbering

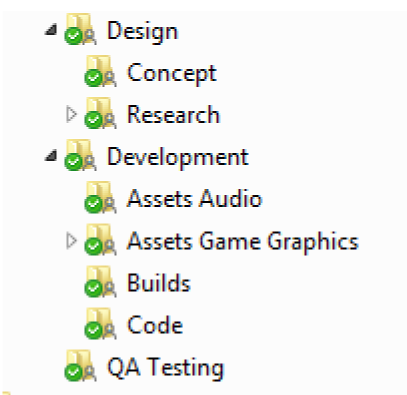
All builds need to use the following sequential numbering system:
1.00 underscore followed by a number sequence

For example:
GameBuild1.00
GameBuild1.02

To note:
- Each new revision is increased by 0.01
- Major updates would be increased by 1 e.g. 1.01 to 2.01
- Avoid special characters such as ~ ! @ # $ % ^ & * ( ) ` ; < > ? , [ ] { } ' " |
- Do not use spaces. Instead, use underscores between words, e.g. file_name

File Organisation

The FF development directory should be used for all projects. Files should be stored in the correct folders and new folders should be used to group similar assets.



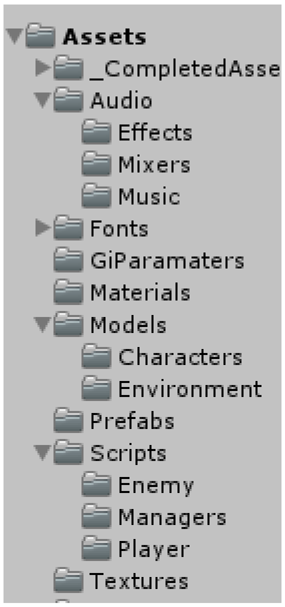
Asset Naming

Assets (graphics, audio, text, etc) should be named according to the FF Asset Naming Convention document.

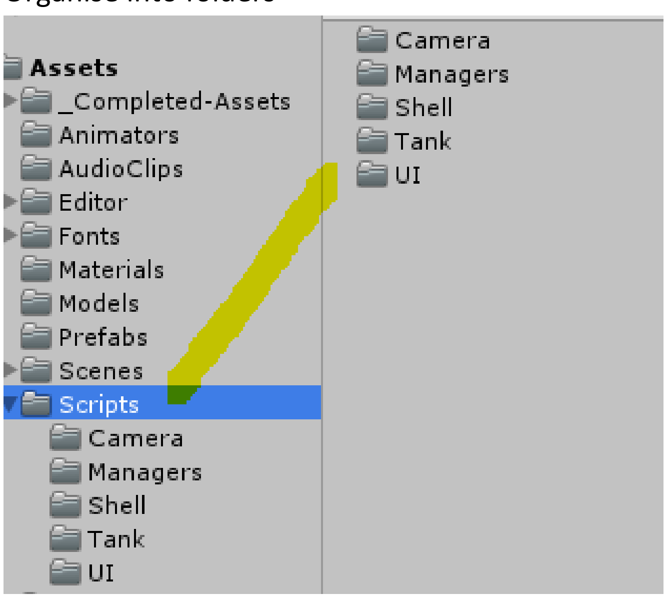
UNITY

C# Coding Conventions
Use the Microsoft standard: https://msdn.microsoft.com/en-us/library/ff926074.aspx

Naming Convention: Scripts and Assets
Project Assets: Logical Folder Structure

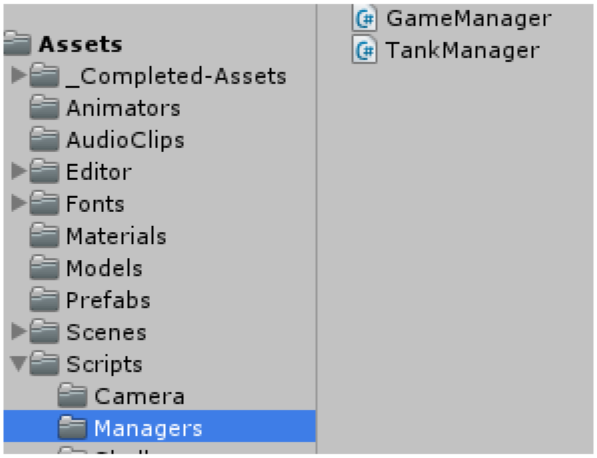


Scripts
Organise into folders

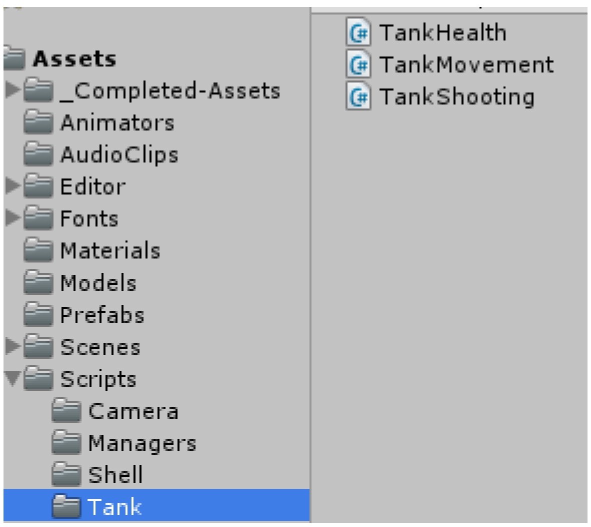


Scripts Names

Managers for groups or systems



Where similar, start with the object name



Script Layout

- All to include Fierce Fun copyright © notice – header comments
- Put public variables first
- All variable should be explicitly declared public or private