TDD Template

Contents

[Variables: 2](#_Toc65922907)

[Public: 2](#_Toc65922908)

[Private: 2](#_Toc65922909)

[Protected: 2](#_Toc65922910)

[Getters and Setters: 2](#_Toc65922911)

[Methods: 2](#_Toc65922912)

[Public: 2](#_Toc65922913)

[Private: 2](#_Toc65922914)

[Protected: 2](#_Toc65922915)

[File Names: 3](#_Toc65922920)

[Scripts: 3](#_Toc65922921)

[Materials: 4](#_Toc65922924)

[Flow Graphs 5](#_Toc65922930)

# Variables:

## Public:

Public variables will use rotating camel case where the first letter of the first word will have a lower case. All words following that word will include a capital letter to start the new word.

public var publicExample;

## Private:

Private variables will use rotating camel case where the first letter of the first word will have a lower case. All words following that word will include a capital letter to start the new word.

private var privateExample;

## Protected:

Protected variables will use the same system as private.

protected var protectedExample;

## Getters and Setters:

Where possible, the Getter and Setter for a private or protected variable will use the same name as the variable, however, it will not have an underscore and the first letter will be a capital letter.

private var \_privateExample;

public var PrivateExample{get{return \_privateExample;}set{\_privateExample = value;]}

# Methods:

All methods will follow the naming convention that the first letter of the first word is capitalized, the first letter of each word following will also be capitalized. Methods do not need special names with the exception to those that return a bool, these should be formed in a question such as IsVisible or CanSee.

## Public:

Same as above, fill if needed.

## Private:

Same as above, fill if needed.

## Protected:

Same as above, fill if needed.

# File Names:

## Duplicate objects:

PineTree (1)

PineTree (2)

PineTree (3)

## Scripts:

Scripts will use a “binomial name” which indicates what this script is and where it derives from. The exception to this is the base class, which will include the term Base in its name. The first letter of each word will be capitalized. For example:

ItemBase

EquipItem

WeaponEquip

MeleeWeapon

Classes that do not have children that inherit from them do no require the Base term.

Any script that will be used as a manager will include the term Manager in its name. For example:

AudioManager

Any scripts that do not fall into these categories are free to have any name as long as it makes sense and is easy to understand.

## Materials:

The given name should reflect what this is a material of, such as wood, grass, etc. Example:

Grass

# Flow Graphs:

Graphs will be laid out so that it is clear where one class starts, what inherits from it, and what variables and methods it will require. Graphs are not written in stone and will rarely include everything required by the end of a project. They act as a starting point to ensure baseline functionality.

Class Names: Blue

Public Elements: Green

Private Elements: Red

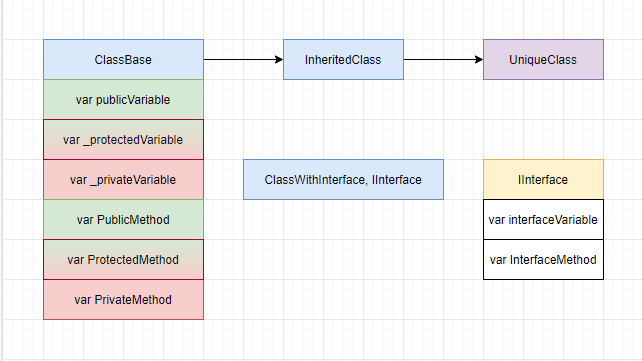
Protected Elements: Gradient of Green to Red

Unique Classes: Purple

Interface Names: Yellow

Interface Variables: White

All previously defined naming conventions will be applied to the table, for example:



Lines will be used to help define the relationship between classes and methods. A solid line that goes from one class to another will indicate a parent:child relationship. A dotted line from one method to another will indicate the methods communicating to each other. These dotted lines should be read from top to bottom to understand the general layout of the code.

