Coding Standards

File Formatting

Indentation

Each level of code should be indented using four spaces and use of tab-spacing should be avoided.

Libraries

Library imports should be ordered alphabetically. The top level library should appear first when importing multiple sub libraries.

Example 1:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Example 2:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
```

Spacing

Developers should follow the below spacing guidelines:

- Place 2 blank lines after library imports
- Place 2 blank lines after the final curly brace of each class
- Place 1 blank line before each function declaration
- Place 1 blank line to separate public member variables from private member variables
- Place additional blank lines to separate distinct sections of code

Naming Conventions

Classes should use pascal case beginning with a capital letter

TestClass

Functions should use camelcase starting with a lowercase letter

testFunction

Variables use camelcase, unless the variable is a single word

testVariable or variable

Constant variables should be all caps with an underscore between words

TEST_CONSTANT

Filenames should have every word capitalized

ThisFile

Documentation

Files

Files comments should be placed at the very beginning of the file before any library imports or code. They should follow the format below.

```
/*
 * File Name
 * Developer's Name
 * Purpose
 */
```

Classes

Class comments should be placed directly before the class declaration. The first line should be a brief summary of the class, followed by the member variables and their purposes.

```
/*
 * an example class to demonstrate comments
 *
 * member variables:
 * classes - number of classes
 * ExampleClass() - constructor
 */
public class ExampleClass
{
    int classes;

    ExampleClass()
    {
        classes = 0;
    }
}
```

Functions

Function comments should be placed directly before the function declaration, following the format below. Include details about return type and parameters if needed.

```
/* function to check if a name starts with an 'A'
 *
 * string parameter of the name
 *
 * returns true if the name starts with A, false otherwise
 */
bool StartsWithA (string name)
{
    if (name[0] == 'A')
    {
        return true;
    }

    return false;
}
```

```
// function to print hello world to ouput
void HelloWorld()
{
    Console.WriteLine("Hello World!");
}
```

Additional Comments

Additional comments should be placed using the "//" comment style. Enter one space between the comment delimiter and the comment text. Use "/* … */" for multiline comments.

```
// check if the first index is an 'A'
if (name[0] == 'A')
```

```
{
    return true; // the first letter is an 'A'
}
```

```
/* if the first letter is not 'A' after
 * checking, then return false
 */
return false;
```

Prefabs

Prefabs should be documented in a README file, and placed in the Doc folder on GIT as a PDF file. The README should be similar to the items you find in the Asset Store on Unity. The file **must**:

- Be well named (If I was looking for your type of item, would the search engine find yours?)
- How to use it (and how to integrate it into your code.)
- Troubleshooting tips if it does not work right

Other Formatting

Files

The following general rules should be followed when formatting your code:

- Max line length should not exceed more than ~120 characters
    - Can be exceeded in cases of long Debog.Log() calls
- Curly braces should be placed on the line after function, class, or other declarations

Classes

- Public variables should appear before private variables in classes
- Public functions should appear before private functions in classes
- Unity functions should appear before custom functions

Error Handling

Exceptions

An exception is defined as an event that occurs during the execution of a program that is unexpected by the program code. In such a case, we create an exception object and call the exception handler code. Exceptions should be used whenever there is the reasonable possibility of a major error occurring. Unity's exception logging syntax should be used to print the error as in the example below:

```
public void MyGameMethod()
{
    try
    {
        // Code that can throw an error
    }
    catch (Exception e)
    {
        // Code to handle the exception here
        Debug.LogException(e, this);
    }

}
```