Coding Standards

—

Naming

Scenes

Town01	❏ Noun ❏ Contains variant number ❏ Use Pascal Case

Scripts

PlayerMovementComponent	❏ Use Pascal Case

Classes

`public class` **`Weapon`** `: MonoBehaviour {}`	❏ Use Pascal Case ❏ Class Names are Nouns

Properties

`public class Weapon : MonoBehaviour {`
`  public int` **`Damage`** `{ get; set; }`
`}`
	❏ Use Pascal Case
	❏ Nouns or Adjectives

Fields

`public class Weapon : MonoBehaviour {`
`  private int` **`_health;`**
`}`
	❏ camelCase
	❏ _underscore
	❏ Nouns or Adjectives

Functions/Methods

`public class Weapon : MonoBehaviour {`
`  public int` **`FireWeapon()`** `{}`
`}`
	❏ Use Pascal Case
	❏ Verbs

Function/Method Parameters

```
public class Player : MonoBehaviour {
      public int TakeDamage(int amount)
      {}
}
```

- ❏ camelCase
- ❏ Nouns or Adjectives

Constants

```
public class Player : MonoBehaviour {
  public const int MAX_PLAYERS = 3;
}
```

- ❏ SCREAMING_CAPS
- ❏ Nouns or Adjectives

Events

```
public class Player : MonoBehaviour {
  public event Action<int> OnTookHit;
}
```

- ❏ Pascal Case
- ❏ 'On Something Happened'

Enums

```
public enum EnumName
{
 ENUM_VALUE = 0,
 ...
}
```

- ❏ PascalCase for enum name
- ❏ SCREAMING_CAPS for enum value

Branches Structure

- **Master** - kept clean and in a working condition, basically our polished stuff go here
  - **Dev** - our dump, here we will merge our individual branches and fix issues and decide what goes into Master
    - **Personal branches** - everyone works on a separate branch on their tasks without interfering with the work of others.

Every time you start work your first job should be to merge the development branch into your own so you get new changes. When finishing work you should commit to the development branchy so others can see your changes and not use an outdated version of the project.

When merging your branch, first merge the development branch and fix any issues that arise there. When there are none, merge your branch into Dev

Abbreviations

- Avoid abbreviations except in cases where it's a domain specific common term.
    - Example: Prefer **playerWeapon** over *pWeap*
    - Exception Example: Prefer **GPU** over *GraphicalProcessingUnit*

- Avoid single character names except as loop iterators with 'i'
    - If multiple iterators are needed, give them proper names
    - Iterate with **++i**, instead of i++

Documentation

- Comments on core mechanics/improvised or unique algorithms. Brief definition of functions/variables.
    - XML style comments for the functions including their params
    - CnC adapted style of documentation for the files - style similar to what CnC used in the 90's for documentation. Example .cpp/ .h

```
/******************************************************************************************
***              C O N F I D E N T I A L  ---  T H E  T E A M  S T U D I O S            ***
******************************************************************************************
*                                                                                        *
*                  Project Name : Fight For Freedom                                      *
*                                                                                        *
*                     File Name : < FILENAME >                                           *
*                                                                                        *
*                    Programmer : < NAME >                                               *
*                                                                                        *
*----------------------------------------------------------------------------------------*
*                                                                                        *
*                   <FUNCTION LIST w/ short description                                  *
*                                                                                        *
*----------------------------------------------------------------------------------------*/
```

Some good practises

- Prefer handling logic using Events over Update
- Make everything private (unless you really need it public to access it in another script), if you want to access it in the Editor use [SerializeField] in front of the object/variable

Serialisable fields show up in the inspector. By default public fields are serialisable. As are any fields you mark with SerialiseField.

- **public** - Show up in inspector and accessible by other scripts
- **[SerialiseField] private** - Show up in inspector, not accessible by other scripts
- **[HideInInspector] public** - Doesn't show in inspector, accessible by other scripts
- **private** - Doesn't show in inspector, not accessible by other scripts