Coding Conventions – Sandstorm Games

Contents

[About 2](#_Toc212452797)

[Naming 3](#_Toc212452798)

[Classes & Structs 3](#_Toc212452799)

[Methods 3](#_Toc212452800)

[Variables 3](#_Toc212452801)

[Documentation 4](#_Toc212452802)

[Examples 5](#_Toc212452803)

[Classes & Structs 5](#_Toc212452804)

[Methods 5](#_Toc212452805)

[Private Method 5](#_Toc212452806)

[Public method 6](#_Toc212452807)

# About

This document outlines the c# coding conventions for Sandstorm Games

# Naming

## Classes & Structs

* All classes and structs should use PascalCase
* Eg. MyClass, MyStruct
* Private classes shall have a leading underscore.
  + Eg. \_MyPrivateClass
* Serializable classes must start with [System.Serializable]

## Methods

* All methods should use PascalCase
* Eg. MyFunction()
* Boolean Methods: Methods returning a bool should be phrased as questions, starting with a verb like Is or Has
* Private methods shall have a leading underscore.
  + Eg. \_MyPrivateFunction

## Variables

* All variables should use camelCase
  + Eg. myVariable
* Private variables should be prefixed with an underscore
  + Eg. \_myPrivateVariable
* Constants should be in ALL\_CAPS separated by underscores
  + Eg. MY\_CONSTANT

# Documentation

Code Formatting and Structure:

* Indentation: Consistent indentation (typically 4 spaces) for readability.
* Braces: Consistent placement of curly braces (e.g., K&R style or Allman style).
* Blank Lines: Use blank lines to separate logical sections of code within methods or between members of a class.
* Comments: Use comments (// for single-line, /\* \*/ for multi-line) to explain complex logic or non-obvious code sections.
* Small, Focused Units: Keep classes and methods relatively small and focused on a single responsibility to improve readability and maintainability.

# Examples

## Classes & Structs

[System.Serializable]

public class NoiseLayer

{

[SerializeField]

[Range(1, 30)]

public float frequency = 5;

[SerializeField]

[Range(0, 1)]

public float intensity = 0.5f;

}

## Methods

### Private Method

void \_ChunkSetup()

{

noiseHandler = FindFirstObjectByType<NoiseHandler>();

nodeMap = new bool[chunkSize + 1, chunkSize + 1];

for (int x = 0; x < chunkSize + 1; x++)

for (int y = 0; y < chunkSize + 1; y++)

nodeMap[x, y] = noiseHandler.NoiseValue(transform.position.x + x, transform.position.y + y) > noiseHandler.noiseThreshold;

}

### Public method

public void DestroyInRadius(Vector3 position, float radius = 30)

{

/\*

\* Destroys all terrain in a radius, based on a point in space.

\*/

Vector2 localPosition = position - transform.position;

bool[,] newNodeMap = new bool[chunkSize + 1, chunkSize + 1];

for (int x = 0; x < chunkSize + 1; x++)

{

for (int y = 0; y < chunkSize + 1; y++)

{

newNodeMap[x, y] = nodeMap[x, y];

if (Vector2.Distance(new Vector2(x, y), localPosition) < radius)

{

newNodeMap[x, y] = false;

}

}

}

// Don't re-render if no nodes have changed

if(newNodeMap != nodeMap)

{

nodeMap = newNodeMap;

\_RenderMesh();

}

}