Documentation

Table of contents

Examples

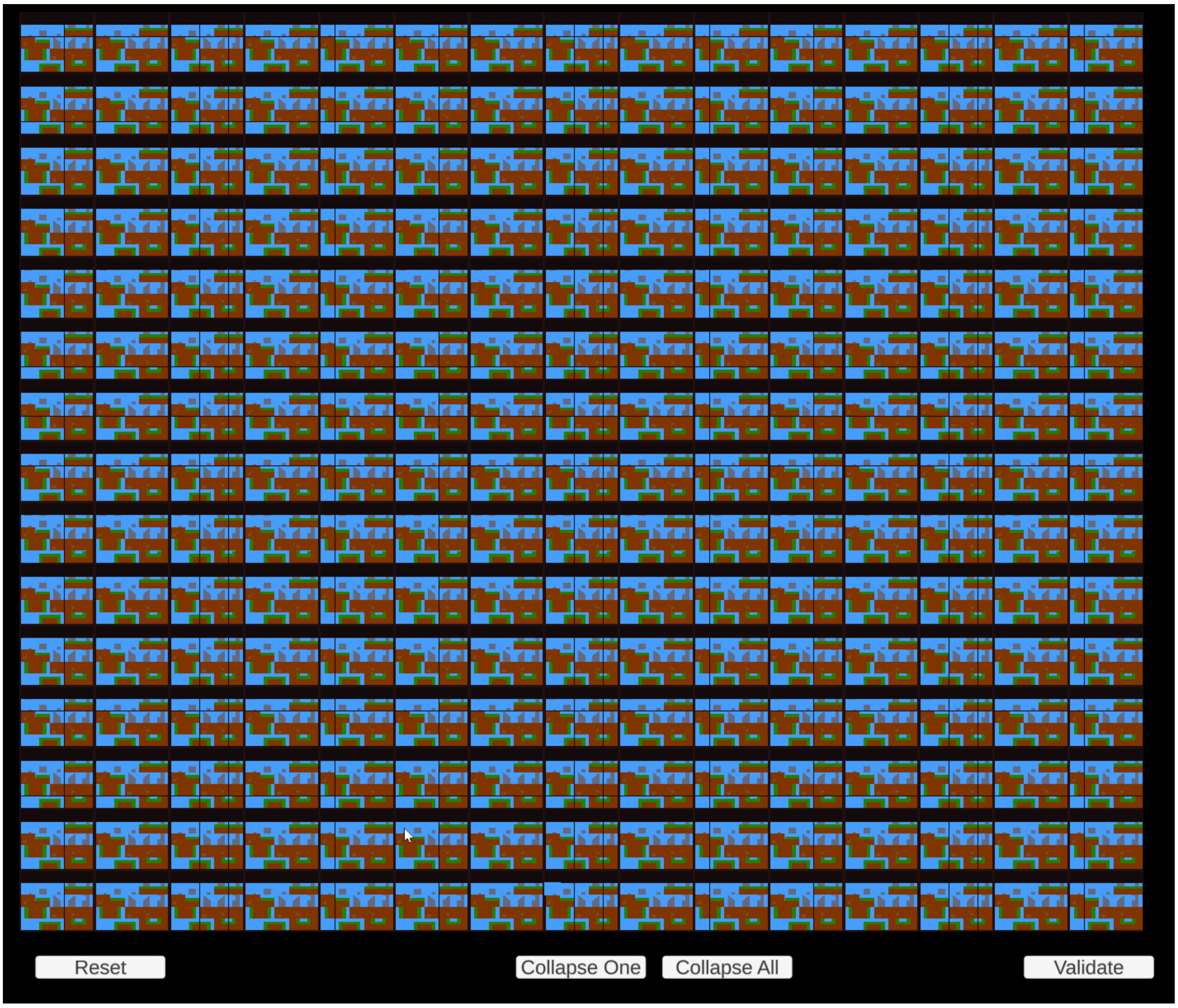
Overview

Code Docs

Example 1 - Color Constraints

A simple world with colors

Watch the Youtube Video



Finding the Example

The Color example can be opened by opening the `Help/Wave Function Collapser` menu item in Unity, and selecting the first available example.

Basic Overview

It can be helpful to identify the WFC primitives in the example. The Slots are each grid cell. The Modules are all the possible sprites that could go into each grid cell. The Constraints are created automatically from the WFC Config Scriptable Object. The Sprite Sheet configuration that is being used in this example uses the colors along the edges of a module's sprite to determine what valid module neighbors can be. If a sprite had a red right edge, it couldn't be placed next to a module with a green left edge.

Configuration Options

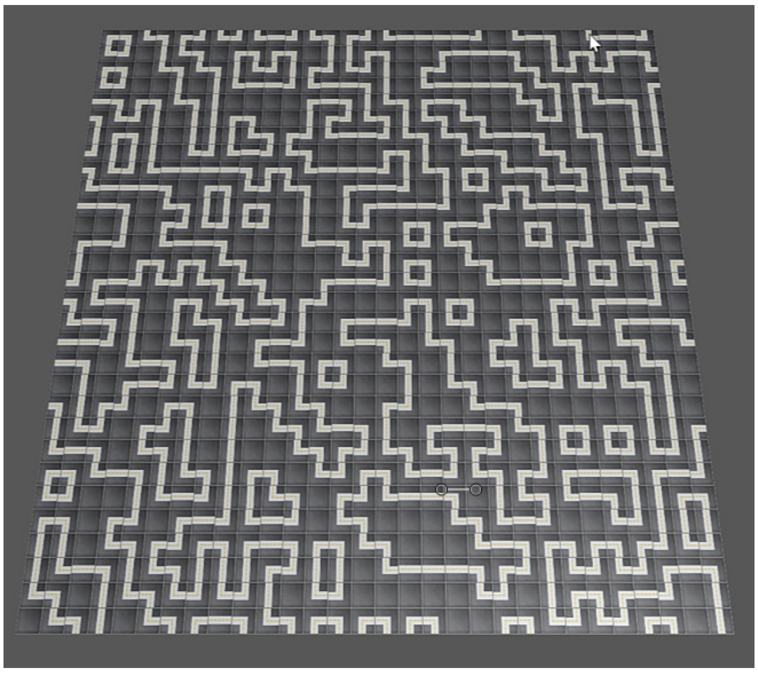
The Sprite Sheet configuration has several configuration options. You can see what each option does by referring to the table below... | Option | Type | Description | | ----------- | ----------- | - | | Use Seed | bool | `true` if every generated Generation Space should use the same seed. | | Seed | int | if `UseSeed` is `true`, what number should be used to seed the random number generator | | Width | int | the x-dimension of the generated Generation Spaces | | Height | int | the y-dimension of the generated Generation Spaces | | Sample Count | int | how many texture samples to take along each edge of each module's sprite | | Sample Spacing Ratio | float | how compressed are the texture samples? | | Sample Spacing Offset | float | are the texture samples offset? | | Sample Tolerance | float | how different can texture color samples be and still be considered equal? | | Sample Left Padding | int | for samples taken for the left edge, how many pixels in from the left edge should the samples be taken? | | Sample Right Padding | int | for samples taken for the right edge, how many pixels in from the right edge should the samples be taken? | | Sample Top Padding | int | for samples taken for the top edge, how many pixels in from the top edge should the samples be taken? | | Sample Low Padding | int | for samples taken for the bottom edge, how many pixels in from the bottom edge should the samples be taken? |

Setting up the GameObject

The main Gameobject in the scene is a `WFCBoard` object. It draws all of the module possibilities per slot. The object is also responsible for handling user interactions, and running the collase.md) operations on the Generation Space You can make your own `WFCBoard` and assign it whatever configuration you'd like by creating one from the Hierarchy window. Right click in the Hierarchy window and select `BrewedInk WFC/2D Grid Renderer`

Example 2 - Socket Constraints

Watch the Youtube Video



Getting Started

Welcome

Wave Function Collapser is a Unity package for solving the Wave Function Collapse algorithm (WFC) with a code first solution. If you are looking for information on how the WFC algorithm works in general, check out these amazing videos by Oskar Stalberg and Martin Donald. If you'd rather do some reading, checkout this overview.

Installing

The code is available on the Unity Asset Store. After you import the package into your project, you should be all set. There is a folder named `com.brewedink.wfc`. Do not rename the folder.

Quick start

If you want to verify things are working, you should check out the example scenes included in the package. You can open the help page by going to `Help/Wave Function Collapser`. See the documentation examples here...

- 1 - Color based Constraints
- 2 - Socket based Constraints

Code First

The hardest thing you'll have to do as you integrate the Wave Function Collasper into your project, is create your own modules and constraints. The asset is meant to solve the WFC algorithm in a general sense, which means you need to provide your own specific implementations of constraints and modules. The Wave Function Collapser comes with a few standard constraints and modules, but you should be ready to create your own.

The entry point to the WFC is the Generation Space class. It is responsible for maintaining all of the slots and modules possibilities as the algorithm runs. When you create a Generation Space, it starts out every slot with all module possibilities. There are methods on the class itself to run the WFC algorithm step by step, or to solve the entire space.

Custom Modules and Constraints

Before you can create a Generation Space, you need to have a set of modules with constraints. You can use whatever you want for modules. You should feel free to subclass the Module class and use those. The only requirement of a subclassed Module, is that you **Must implement the HashCode and Equals method**

The Generation Space

You can create a Generation Space with the utility method, From2DGrid.md)

```
var space = GenerationSpace.From2DGrid(
 3,       // the width of the grid
 3,       // the height of the grid
 modules, // the modules that every slot will start with
 seed:24  // an optional random seed that controls how random collapses happen
);
```

Or you can create a Generation Space by hand using the constructor.md).

```
var space = new GenerationSpace(
 slots,   // all of the slots in the generation space. Similar to nodes in a directed graph.
 edges,   // all of the edges that connect slots. Similar to edges in a directed graph.
 modules, // the modules that every slot will start with
 seed:24  // an optional random seed that controls how random collapses happen
);
```

Running the Algorithm

Once you have a Generation Space set up, you can run the WFC algorithm on it. There are a few ways to run the algorithm but they all generally work the same way. You can either collapse a single Slot at a time, which may be helpful to author constraints onto the outcome, or you can collapse the entire space at once. The simplest use case is to collapse the entire space all at once.

All of the methods that collapse the wave function return enumerable sets of WFCProgress. The methods use the C# generator pattern. Each instance of WFCProgress will tell you about the most recent action taken in the WFC algorithm. You can inspect the instance to see if a Module was removed from a Slot's superposition, or if an error happened, or if the algorithm is just continuing to run. When you call any `Collapse` method on the Generation Space, the operation won't be completed until you completely iterate through the resulting `IEnumerable<WFCProgress>`. An easy way to do that is with a `foreach` loop. This also provides you an easy way to perform the algorithm at a rate that fits your desired framerate. In the code sample below, the entire Generation Space is collapsed all at once, with no render frames or pausing. Depending on the size of the Generation Space, this could cause a visible lagtime in the game.

```
var collapseOperation = space.Collapse();
foreach (var progress in collapseOperation)
{
   // you can inspect the progress variable, or yield a render frame, or immediately proceed to the next progress operation.
   switch (progress) {
       case SlotModuleRemoved removed:
            break; // a module was removed from a Slot's superposition
       case WFCError error:
            break; // something went wrong
       default:
            break; // a non-interesting progress event. The algorithm is working hard...
   }
}
```

A better way to run the WFC algorithm would be to run it inside a Unity coroutine. In the sample code below, every progress element is interleaved with a render frame. This means that the game won't lag or pause, but that the algorithm will take much longer in realtime to finish.

```
public class ExampleBehaviour : MonoBehaviour {

   public GenerationSpace space;

   void Start() {
       StartCoroutine(RunWFC());
   }

   IEnumerator RunWFC() {
       foreach (var _ in space.Collapse()){
            yield return new WaitForEndOfFrame();
       }
   }
}
```

However, spinning up and managing your own `Coroutine` every time time you want to invoke a collapse operation can be tedious. There is a utility method that will let you run a collapse operation in a `Coroutine`, control how much time per frame is allocated to the algorithm, attach callbacks to various events, and inspect the state of the operation asynchronously. In the example below, the RunAsCoroutine.md) method converts a set of `IEnumerable<WFCProgress>` into a WFCProgressObserver

```
public class Example : MonoBehaviour {

   public GenerationSpace space;
   private WFCProgressObserver _handle;

   void Start() {
        _handle = space.Collapse()
            .RunAsCoroutine(this, frameBudgetTime)
            .OnSelectedModule<SpriteConfigModule>((slot, module) =>
            {
                // a module of `SpriteConfigModule` has been selected
            })
            .OnCompleted(() =>
            {
                // the operation is complete.
            });
   }

    void Update()
    {
        if (_handle.IsComplete)
        {
            // the operation is complete
            var selections = _handle.SlotSelections.Count;
            var operations = _handle.ProgressCount;
        }
    }
}
```

Wave Function Collapse

The purpose of the Wave Function Collapse (WFC) algorithm is to procedurally solve problems that could have many solutions. The algorithm was popularized in this Github Project by mxgmn.

There are already several fantastic explanations of the WFC algorithm.

- Robert Heaton

Quick Terms

Slot

A Slot is a physical spot in the WFC. When the WFC starts, each Slot has many Module possibilities. When the WFC ends, each Slot has one possible Module. For example, in a 2D world generation demo, each cell in the 2D grid is a Slot.

Module

A Module is a potential outcome for a Slot. Modules are placed on Slots. For example, in a 2D world generation demo, a Module could be a tile of grass, a tile of water, a tile of sky, or any other world piece. A Module has a set of constraints about how it can be placed. When a constraint is invalidated for a Slot, the Module becomes invalid for the Slot.

Superposition

The superposition is the set of possible Modules per Slot. As the WFC algorithm runs, modules are removed as possibilities from each Slot, and the superposition shrinks. Eventually, the superposition completely collapses such that there is only one possible Module per Slot.

Slot Entropy

Entropy is a rough measure of how many Modules are possible per Slot. At the start of the WFC, the Entropy of each Slot is high, because all Modules are possible per Slot. As the algorithm runs, the Entropy lowers. The Entropy is similar to the amplitude of the superposition. The entropy of a slot is also related to the weights of the modules still available at that slot. Modules with higher weights produce lower entropies.

Module Constraints

Each Module has a set of Constraints that control how the Module can be placed on Slots. A common example is an Adjacency Constraint. A Module, X, may have an Adjacency Constraint that says, "only Modules A, B, or C can exist above this Module". If a Slot doesn't have Modules A, B, or C in its superposition, then the Slot below that cannot contain Module X, because the Adjacency constraint isn't valid.

AdjacencyConstraint

brewkedink.wfc

BrewedInk.WFC

AdjacencyConstraint Class

A WFC constraint that enforces neighboring modules per slot.
For example,
This constraint lets a module express that its western neighbor must be of a certain module.

```
public class AdjacencyConstraint : BrewedInk.WFC.ModuleConstraint
```

Inheritance System.Object ModuleConstraint AdjacencyConstraint

Fields

Delta

The positional difference this constraint applies for. If the module is being tested between two slots (A, and B), the constraint only applies if the positional difference between A and B equals the Delta.

NeedsOneOf

A set of possible module values that can exist at a slot with the given Delta.

AdjacencyConstraint.Delta

brewkedink.wfc

BrewedInk.WFC.AdjacencyConstraint

AdjacencyConstraint.Delta Field

The positional difference this constraint applies for. If the module is being tested between two slots (A, and B), the constraint only applies if the positional difference between A and B equals the Delta.

```
public Vector3Int Delta;
```

Field Value

UnityEngine.Vector3Int

AdjacencyConstraint.NeedsOneOf

brewkedink.wfc

BrewedInk.WFC.AdjacencyConstraint

AdjacencyConstraint.NeedsOneOf Field

A set of possible module values that can exist at a slot with the given Delta.

```
public ModuleSet NeedsOneOf;
```

Field Value

ModuleSet

BrewedInk.WFC

brewkedink.wfc

BrewedInk.WFC Namespace

Classes

AdjacencyConstraint

A WFC constraint that enforces neighboring modules per slot.
For example,
This constraint lets a module express that its western neighbor must be of a certain module.

GenerationSpace

The Generation Space is the main access point for the WFC algorithm. The Generation Space holds a directed graph of Slots, and a set of all possible modules per slot.
The Generation Space has methods for collapsing the superposition of the Slots, until all slots only have one module option left.

Module

In the WFC algorithm, a module is a possibility for some Slot

ModuleConstraint

The base type for all WFC constraints. Feel free to subclass this type and make your own constraints.

ModuleSet

A ModuleSet is a collection of modules. It is a HashSet, with a few helpful methods attached to it.

ModuleSet<TModule>

A ModuleSet is a collection of modules. It is a HashSet, with a few helpful methods attached to it.
The generic type in the hash set must be assignable from the Module type

Slot

In the WFC algorithm, a Slot is a place that holds one Module. As the Wave Function is being collapsed, the slot may have many potential modules available to it.
As the collapse continues, modules are removed as possibilities from the slot, until there is only one module possible for the slot.

SlotCannotHaveEmptyModuleSetException

An exception that is thrown when the WFC tries to remove the last remaining module from a slot's possibility space. A slot cannot have zero possibilities.

SlotCannotSelectUnavailableModule

An exception that is thrown if a slot tries to select a module that is no longer possible, due to constraints.

SlotEdge

A SlotEdge is the connection between two slots. In a simple 2D 1X2 grid case, there are two slots, and one slot edge connecting them.
The term "edge" comes from Graph Theory. These edges are directional.

SlotModuleRemoved

A progress instance that explains what module was removed from a slot's possibility space.

SlotModuleSelected

A progress instance that explains what module was selected for a slot. When this progress type shows up, it means the superposition for the given slot has collapsed to the given module.

WCFConfigObject

A ScriptableObject wrapper that streamlines creating GenerationSpace instances from MonoBehaviours.

WCFConfigObject<TModuleObject,TModule>

WFCError

A WFCError instance could be returned by the WFC algorithm. If you receive one of these instances, it means something with the process has broken.

WFCGridRenderer

A wrapper class for displaying a helpful 2D visual for a WFC run.

WFCProgress

The WFCProgress class represents a unit of progress in the WFC algorithm. As the algorithm is run, a sequence of WCFProgress items will be returned.
Different types of progress will be returned, and you can switch on the type to gain insights to the algorithm's decisions...

WFCProgressExtensions

A Utility class for IEnumerable sets of WFCProgress

WFCProgressObserver

A wrapper class around the invocation of a WFC collapse operation.
You should use this in conjunction with RunAsCoroutine(IEnumerable<WFCProgress>, MonoBehaviour, float).md
'BrewedInk.WFC.WFCProgressExtensions.RunAsCoroutine(System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress>, UnityEngine.MonoBehaviour, float)')

GenerationSpace

brewkedink.wfc

BrewedInk.WFC

GenerationSpace Class

The Generation Space is the main access point for the WFC algorithm. The Generation Space holds a directed graph of Slots, and a set of all possible modules per slot.
The Generation Space has methods for collapsing the superposition of the Slots, until all slots only have one module option left.

```
public class GenerationSpace
```

Inheritance System.Object GenerationSpace

Constructors

GenerationSpace(List<Slot>, List<SlotEdge>, ModuleSet, Nullable<int>).md 'BrewedInk.WFC.GenerationSpace.GenerationSpace(System.Collections.Generic.List<BrewedInk.WFC.Slot>, System.Collections.Generic.List<BrewedInk.WFC.SlotEdge>, BrewedInk.WFC.ModuleSet, System.Nullable<int>)')

Create a new instance of the WFC algorithm. Each time you construct an instance of the Generation Space, you are setting up a new superposition with nothing collapsed.
You can also use utility methods to create a new Generation space. From2DGrid(int, int, ModuleSet, Nullable<int>, Action<List<Slot>,List<SlotEdge>>).md 'BrewedInk.WFC.GenerationSpace.From2DGrid(int, int, BrewedInk.WFC.ModuleSet, System.Nullable<int>, System.Action<System.Collections.Generic.List<BrewedInk.WFC.Slot>,System.Collections.Generic.List<BrewedInk.WFC.SlotEdge>>)')

Methods

Collapse().md 'BrewedInk.WFC.GenerationSpace.Collapse()')

Run the WFC algorithm to completion on the Generation Space.
This method will keep on collapsing the slot with the lowest entropy until all slots have been collapsed to a single module possibility.

This method may take a long time to complete if the Generation Space is large. You should interweave the resulting progress set with a Coroutine with render frames.

CollapseLowestEntropySlot().md 'BrewedInk.WFC.GenerationSpace.CollapseLowestEntropySlot()')

Collapse the superposition of some slot. The slot that is selected is the slot in the Generation Space with the lowest entropy. A Slot's entropy is found by taking the count of available modules remaining at the slot.
This will cause many propagation events, and may cause other slots to collapse to a single module possibility as well.
By default, WFC always prefer to collapse the slot with the lowest entropy, because it is less likely to produce an invalid outcome.

If there are multiple slots with the same entropy, the first one detected in the Generation Space's internal data structure will be used. It is not random. It depends on how the slots were created in the constructor of the Generation Space.

CollapseSlot(Slot, Module).md 'BrewedInk.WFC.GenerationSpace.CollapseSlot(BrewedInk.WFC.Slot, BrewedInk.WFC.Module)')

Collapse a slot to a given module outcome.
If a slot had N possible modules, after this method runs, the slot would only have 1 possible module.
This will cause many propagation events, and may cause other slots to collapse to a single module possibility as well.

CollapseSlot(Slot).md 'BrewedInk.WFC.GenerationSpace.CollapseSlot(BrewedInk.WFC.Slot)')

Given a slot with many module possibilities, collapse the possibility space so that only one module is possible for the given slot.
This will cause many propagation events, and may cause other slots to collapse to a single module possibility as well.
If the given slot has many module possibilities, one module will be selected at random.

From2DGrid(int, int, ModuleSet, Nullable<int>, Action<List<Slot>,List<SlotEdge>>).md 'BrewedInk.WFC.GenerationSpace.From2DGrid(int, int, BrewedInk.WFC.ModuleSet, System.Nullable<int>, System.Action<System.Collections.Generic.List<BrewedInk.WFC.Slot>,System.Collections.Generic.List<BrewedInk.WFC.SlotEdge>>)')

A utility method for building a Generation Space in the form of a 2D grid.
All Generation Spaces are directed graphs, and the resulting Generation Space from this method will be a graph representing a 2D grid. Each cell in the grid will have neighbors to the west, east, south, and north. There are no diagonal neighbors.

GetEdges(Slot).md 'BrewedInk.WFC.GenerationSpace.GetEdges(BrewedInk.WFC.Slot)')

Get a Slot's edges.
SlotEdge

GetEntropy(ModuleSet).md 'BrewedInk.WFC.GenerationSpace.GetEntropy(BrewedInk.WFC.ModuleSet)')

Calculates the entropy of a set of modules.
Entropy is `csharp -sum(p log(p) )` over all modules where each p is the probability of the module

GetEntropy(Slot).md 'BrewedInk.WFC.GenerationSpace.GetEntropy(BrewedInk.WFC.Slot)')

Calculates the current entropy of a slot.
Entropy is roughly equal to the number of modules left in the slot's wave function. The contribution of each module is weighted by the module's weight. Weight
The exact entropy calculation is `csharp -sum( p log(p) )` over all modules where each p is the probability of the module.

GetSlot(Vector3Int).md 'BrewedInk.WFC.GenerationSpace.GetSlot(UnityEngine.Vector3Int)')

Find a slot given a position. If no slot exists at the given position, this method will return null.

GetSlotOptions(Slot).md 'BrewedInk.WFC.GenerationSpace.GetSlotOptions(BrewedInk.WFC.Slot)')

At any given moment, a slot may have many possible modules available to it. This method will give you the set of possible modules available on a slot.

RemoveSlotOption(Slot, Module).md 'BrewedInk.WFC.GenerationSpace.RemoveSlotOption(BrewedInk.WFC.Slot, BrewedInk.WFC.Module)')

A small way to collapse the wave function. At some given slot, this method removes some given module from the set of possible modules at the slot.
Before this method is run, the given slot may have N possible modules. After the method is run, that slot will have N-1 possible modules.
This method collapses the superposition at the given slot, which also means that neighboring slots may need to remove modules given the new position.
This method triggers a propagation event in the Generation Space. As such, it returns a consumable sequence of WFCProgress elements.
The faster you consume the sequence, the faster the entire propagation phase happens. You may wish to sow render frames between calls. The sequence must be iterated through, or the propagation won't finish.

Reset().md 'BrewedInk.WFC.GenerationSpace.Reset()')

Reset all slot superpositions so that all modules are likely again.

TryGetOnlyOption(Slot, Module).md 'BrewedInk.WFC.GenerationSpace.TryGetOnlyOption(BrewedInk.WFC.Slot, BrewedInk.WFC.Module)')

At any given moment, a slot may have many possible modules available to it. However, eventually, a slot will only have one possible module remaining.
This method will help you identify when there is only one module left per slot.
If there is only one module left for the given slot, this method will return true and store the assigned module in the out parameter. If there is more than one module possible, the method will return false, and the out variable will be null.

You can also retrieve all of the available slot modules with the `GenerationSpace.GetSlotOptions` method.
GetSlotOptions(Slot).md 'BrewedInk.WFC.GenerationSpace.GetSlotOptions(BrewedInk.WFC.Slot)')

Validate().md 'BrewedInk.WFC.GenerationSpace.Validate()')

Evaluate all constraints in the Generation Space and return the number if invalid constraints.
Ideally, this method should always return zero. However, you can use this to check that the constraints are indeed valid if
you suspect that the collapse has errored.

GenerationSpace.Collapse()

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.Collapse() Method

Run the WFC algorithm to completion on the Generation Space.
This method will keep on collapsing the slot with the lowest entropy until all slots have been collapsed to a single module possibility.

This method may take a long time to complete if the Generation Space is large. You should interweave the resulting progress set with a Coroutine with render frames.

```
public System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress> Collapse();
```

Returns

System.Collections.Generic.IEnumerable<WFCProgress>
A sequence of WFCProgress representing the completion of the collapse operation. The entire sequence must be enumerated or the collapse won't finish.

GenerationSpace.CollapseLowestEntropySlot()

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.CollapseLowestEntropySlot() Method

Collapse the superposition of some slot. The slot that is selected is the slot in the Generation Space with the lowest entropy. A Slot's entropy is found by taking the count of available modules remaining at the slot.
This will cause many propagation events, and may cause other slots to collapse to a single module possibility as well.
By default, WFC always prefer to collapse the slot with the lowest entropy, because it is less likely to produce an invalid outcome.

If there are multiple slots with the same entropy, the first one detected in the Generation Space's internal data structure will be used. It is not random. It depends on how the slots were created in the constructor of the Generation Space.

```
public System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress> CollapseLowestEntropySlot();
```

Returns

System.Collections.Generic.IEnumerable<WFCProgress>

GenerationSpace.CollapseSlot(Slot)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.CollapseSlot(Slot) Method

Given a slot with many module possibilities, collapse the possibility space so that only one module is possible for the given slot.
This will cause many propagation events, and may cause other slots to collapse to a single module possibility as well.
If the given slot has many module possibilities, one module will be selected at random.

```
public System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress> CollapseSlot(BrewedInk.WFC.Slot slot);
```

Parameters

`slot` Slot
The slot to collapse to a single module possibility

Returns

System.Collections.Generic.IEnumerable<WFCProgress>
A sequence of WFCProgress representing the completion of the collapse operation. The resulting sequence must be enumerated, or the collapse won't complete.

GenerationSpace.CollapseSlot(Slot.Module)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.CollapseSlot(Slot, Module) Method

Collapse a slot to a given module outcome.
If a slot had N possible modules, after this method runs, the slot would only have 1 possible module.
This will cause many propagation events, and may cause other slots to collapse to a single module possibility as well.

```
public System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress> CollapseSlot(BrewedInk.WFC.Slot slot, BrewedInk.WFC.Module module);
```

Parameters

`slot` Slot
A slot to collapse to one module possibility

`module` Module
the module that will be only remaining possibility for the given slot

Returns

System.Collections.Generic.IEnumerable<WFCProgress>
A sequence of WFCProgress representing the progress of the collapse operation. The sequence must be enumerated, or the collapse won't finish.

Exceptions

SlotCannotSelectUnavailableModule
You cannot select a module for a slot if the slot doesn't already contain the module in its superposition.

GenerationSpace.From2DGrid(int.int.ModuleSet.Nullable.int..Action.List.Slot..List.SlotEdge..)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.From2DGrid(int, int, ModuleSet, Nullable<int>, Action<List<Slot>,List<SlotEdge>>) Method

A utility method for building a Generation Space in the form of a 2D grid.
All Generation Spaces are directed graphs, and the resulting Generation Space from this method will be a graph representing a 2D grid. Each cell in the grid will have neighbors to the west, east, south, and north. There are no diagonal neighbors.

```
public static BrewedInk.WFC.GenerationSpace From2DGrid(int width, int height, BrewedInk.WFC.ModuleSet modules, System.Nullable<int> seed=null, System.Action<System.Collections.Generic.List<BrewedInk.WFC.Sl
ot>,System.Collections.Generic.List<BrewedInk.WFC.SlotEdge>> prepareSpace=null);
```

Parameters

`width` System.Int32
The width of the resulting 2D grid Generation Space

`height` System.Int32
The height of the resulting 2D grid Generation Space

`modules` ModuleSet
The set of all possible modules, with constraints prepopulated. Every Slot that is created in this Generation Space will have all modules available at the start of the WFC.

`seed` System.Nullable<System.Int32>
An optional seed that controls how the WFC decides random operations.

`prepareSpace` System.Action<System.Collections.Generic.List<Slot>,System.Collections.Generic.List<SlotEdge>>
An optional callback to control how edges are created in the 2D grid. If this parameter is left null, every slot will have edges to the west, east, south, and north. If you need to change how the edges are created, remove certain edges, or add additional edges, you can provide your own implementation of this callback. If you provide a callback value, you are responsible for populating the entire List of SlotEdges per given Slot.

Returns

GenerationSpace
A Generation Space representing a 2D grid of slots where every slot has every module available to it at the start of the WFC.

GenerationSpace.GenerationSpace(List.Slot..List.SlotEdge..ModuleSet.Nullable.int.)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.GenerationSpace(List<Slot>, List<SlotEdge>, ModuleSet, Nullable<int>) Constructor

Create a new instance of the WFC algorithm. Each time you construct an instance of the Generation Space, you are setting up a new superposition with nothing collapsed.
You can also use utility methods to create a new Generation space. From2DGrid(int, int, ModuleSet, Nullable<int>, Action<List<Slot>,List<SlotEdge>>).md 'BrewedInk.WFC.GenerationSpace.From2DGrid(int, int, BrewedInk.WFC.ModuleSet, System.Nullable<int>, System.Action<System.Collections.Generic.List<BrewedInk.WFC.Slot>,System.Collections.Generic.List<BrewedInk.WFC.SlotEdge>>)')

```
public GenerationSpace(System.Collections.Generic.List<BrewedInk.WFC.Slot> slots, System.Collections.Generic.List<BrewedInk.WFC.SlotEdge> edges, BrewedInk.WFC.ModuleSet allModules, System.Nullable<int> seed);
```

Parameters

`slots` System.Collections.Generic.List<Slot>
A set of slots to perform the WFC on. Every slot will be assumed to have every module as a possibility

`edges` System.Collections.Generic.List<SlotEdge>
A set of edges that connect the slots.

`allModules` ModuleSet
A set of all the modules, with the constraints already provided

`seed` System.Nullable<System.Int32>
An optional random seed. If you provide the same seed, the WFC algorithm will produce the same output given the same inputs.

GenerationSpace.GetEdges(Slot)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.GetEdges(Slot) Method

Get a Slot's edges.
SlotEdge

```
public System.Collections.Generic.List<BrewedInk.WFC.SlotEdge> GetEdges(BrewedInk.WFC.Slot slot);
```

Parameters

`slot` Slot
The slot to get the edges for.

Returns

System.Collections.Generic.List<SlotEdge>
A list of edges for the given slot

GenerationSpace.GetEntropy(ModuleSet)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.GetEntropy(ModuleSet) Method

Calculates the entropy of a set of modules.
Entropy is `csharp -sum(p log(p) )` over all modules where each p is the probability of the module

```
public float GetEntropy(BrewedInk.WFC.ModuleSet modules);
```

Parameters

`modules` ModuleSet
A set of modules

Returns

System.Single
The entropy of the given set of modules

GenerationSpace.GetEntropy(Slot)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.GetEntropy(Slot) Method

Calculates the current entropy of a slot.
Entropy is roughly equal to the number of modules left in the slot's wave function. The contribution of each module is weighted by the module's weight. Weight
The exact entropy calculation is `csharp -sum( p log(p) )` over all modules where each p is the probability of the module.

```
public float GetEntropy(BrewedInk.WFC.Slot slot);
```

Parameters

`slot` Slot
The slot to find the entropy of

Returns

System.Single
The current entropy of the slot

GenerationSpace.GetSlot(Vector3Int)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.GetSlot(Vector3Int) Method

Find a slot given a position. If no slot exists at the given position, this method will return null.

```
public BrewedInk.WFC.Slot GetSlot(UnityEngine.Vector3Int coordinate);
```

Parameters

`coordinate` UnityEngine.Vector3Int

Returns

Slot
The slot at the given position, or null if none exists

GenerationSpace.GetSlotOptions(Slot)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.GetSlotOptions(Slot) Method

At any given moment, a slot may have many possible modules available to it. This method will give you the set of possible modules available on a slot.

```
public BrewedInk.WFC.ModuleSet GetSlotOptions(BrewedInk.WFC.Slot slot);
```

Parameters

`slot` Slot
The slot whose available modules will be returned

Returns

ModuleSet
A set of modules representing all possible modules for the given slot

GenerationSpace.RemoveSlotOption(Slot.Module)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.RemoveSlotOption(Slot, Module) Method

A small way to collapse the wave function. At some given slot, this method removes some given module from the set of possible modules at the slot.
Before this method is run, the given slot may have N possible modules. After the method is run, that slot will have N-1 possible modules.
This method collapses the superposition at the given slot, which also means that neighboring slots may need to remove modules given the new position.
This method triggers a propagation event in the Generation Space. As such, it returns a consumable sequence of WFCProgress elements.
The faster you consume the sequence, the faster the entire propagation phase happens. You may wish to sow render frames between calls. The sequence must be iterated through, or the propagation won't finish.

```
public System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress> RemoveSlotOption(BrewedInk.WFC.Slot startingSlot, BrewedInk.WFC.Module removeModule);
```

Parameters

`startingSlot` Slot
A slot to remove a module option from

`removeModule` Module
The module to be removed from the slot

Returns

System.Collections.Generic.IEnumerable<WFCProgress>
A sequence of WFCProgress representing the progress of the propagation event. The sequence must be iterated through, or the propagation won't occur.

Exceptions

SlotCannotHaveEmptyModuleSetException
An exception will be thrown if removing a module from a slot would cause the slot to have zero possible modules. A slot must always have at least one possibility.

GenerationSpace.Reset()

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.Reset() Method

Reset all slot superpositions so that all modules are likely again.

```
public void Reset();
```

GenerationSpace.TryGetOnlyOption(Slot.Module)

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.TryGetOnlyOption(Slot, Module) Method

At any given moment, a slot may have many possible modules available to it. However, eventually, a slot will only have one possible module remaining.
This method will help you identify when there is only one module left per slot.
If there is only one module left for the given slot, this method will return true and store the assigned module in the out parameter. If there is more than one module possible, the method will return false, and the out variable will be null.

You can also retrieve all of the available slot modules with the `GenerationSpace.GetSlotOptions` method.
GetSlotOptions(Slot).md 'BrewedInk.WFC.GenerationSpace.GetSlotOptions(BrewedInk.WFC.Slot)')

```
public bool TryGetOnlyOption(BrewedInk.WFC.Slot slot, out BrewedInk.WFC.Module module);
```

Parameters

`slot` Slot
The slot whose only remaining available module will be checked.

`module` Module
An out variable for the only remaining available module. After the method is invoked, the value of the module will be the last module for the slot, or null if there are more than one modules remaining.

Returns

System.Boolean
true if there was only one module available, or false otherwise.

GenerationSpace.Validate()

brewkedink.wfc

BrewedInk.WFC.GenerationSpace

GenerationSpace.Validate() Method

Evaluate all constraints in the Generation Space and return the number if invalid constraints.
Ideally, this method should always return zero. However, you can use this to check that the constraints are indeed valid if
you suspect that the collapse has errored.

```
public int Validate();
```

Returns

System.Int32
The number of broken constraints in the Generation Space.

index

brewkedink.wfc

Namespaces

BrewedInk.WFC

Module

brewkedink.wfc

BrewedInk.WFC

Module Class

In the WFC algorithm, a module is a possibility for some Slot

```
public class Module
```

Inheritance System.Object Module

Fields

Constraints

Every module has a set of ModuleConstraint. At the start of the WFC algorithm, every module is possible in all slots.
The constraints for the module in each slot are all still plausible. As the Wave Function collapses, module possibilities are removed, and various constraints become impossible to fulfill. When that happens, this module becomes invalid itself.

Display

Every module must have a unique display name. This name is used for hashing and equality checks.

Weight

A module's weight controls how likely it is to be chosen randomly during a slot collapse. A module with a with a weight of zero will be never be randomly picked.

ModuleConstraint

brewkedink.wfc

BrewedInk.WFC

ModuleConstraint Class

The base type for all WFC constraints. Feel free to subclass this type and make your own constraints.

```
public abstract class ModuleConstraint
```

Inheritance System.Object ModuleConstraint

Derived
↳ AdjacencyConstraint

Methods

ShouldRemoveModule(SlotEdge, GenerationSpace, Module, ModuleSet).md 'BrewedInk.WFC.ModuleConstraint.ShouldRemoveModule(BrewedInk.WFC.SlotEdge, BrewedInk.WFC.GenerationSpace, BrewedInk.WFC.Module, BrewedInk.WFC.ModuleSet)')

The only required method of a WFC constraint.
Given some edge and some module-in-question, should the module-in-question be removed from the edge's source's possibility space?

When the WFC algorithm propagates information, every possible module per slot is checked against every neighboring slot. If ANY neighboring slot has a constraint that disqualifies the module-in-question, the module will be removed from the source's superposition. This in turn, causes a new propagation wave.

ModuleConstraint.ShouldRemoveModule(SlotEdge.GenerationSpace.Module.ModuleSet)

brewkedink.wfc

BrewedInk.WFC.ModuleConstraint

ModuleConstraint.ShouldRemoveModule(SlotEdge, GenerationSpace, Module, ModuleSet) Method

The only required method of a WFC constraint.
Given some edge and some module-in-question, should the module-in-question be removed from the edge's source's possibility space?

When the WFC algorithm propagates information, every possible module per slot is checked against every neighboring slot. If ANY neighboring slot has a constraint that disqualifies the module-in-question, the module will be removed from the source's superposition. This in turn, causes a new propagation wave.

```
public abstract bool ShouldRemoveModule(BrewedInk.WFC.SlotEdge edge, BrewedInk.WFC.GenerationSpace space, BrewedInk.WFC.Module module, BrewedInk.WFC.ModuleSet modulesToRemove);
```

Parameters

`edge` SlotEdge
The edge whose source slot's superposition is being checked. If the method returns true, the given module will be removed from the edge's source's slot's superposition.

`space` GenerationSpace
The generation space that this edge and module are apart of. The space is made available so that checks can be made for other available modules at the edge source and destination slots.

`module` Module
The module-in-question. Should this module be removed from the edge source, because of the edge destination?

`modulesToRemove` ModuleSet
An advanced capability. Anything in this set will be removed from the edge source's superposition. You can manually add to the set, but only if you are extremely confident.

Returns

System.Boolean
Return true if the module should be removed from the edge's source's super position, because the constraint is invalid. Return false if the module doesn't break the constraint.

ModuleSet

brewkedink.wfc

BrewedInk.WFC

ModuleSet Class

A ModuleSet is a collection of modules. It is a HashSet, with a few helpful methods attached to it.

```
public class ModuleSet : BrewedInk.WFC.ModuleSet<BrewedInk.WFC.Module>
```

Inheritance System.Object □ System.Collections.Generic.HashSet<Module> □ BrewedInk.WFC.ModuleSet<Module> □ ModuleSet

ModuleSet.TModule.

brewkedink.wfc

BrewedInk.WFC

ModuleSet<TModule> Class

A ModuleSet is a collection of modules. It is a HashSet, with a few helpful methods attached to it.
The generic type in the hash set must be assignable from the Module type

```
public class ModuleSet<TModule> : System.Collections.Generic.HashSet<TModule>
    where TModule : BrewedInk.WFC.Module
```

Type parameters

`TModule`

Inheritance System.Object 　 System.Collections.Generic.HashSet<TModule> 　 ModuleSet<TModule>

Derived
↳ ModuleSet

Module.Constraints

brewkedink.wfc

BrewedInk.WFC.Module

Module.Constraints Field

Every module has a set of ModuleConstraint. At the start of the WFC algorithm, every module is possible in all slots.
The constraints for the module in each slot are all still plausible. As the Wave Function collapses, module possibilities are removed, and various constraints become impossible to fulfill. When that happens, this module becomes invalid itself.

```
public List<ModuleConstraint> Constraints;
```

Field Value

System.Collections.Generic.List<ModuleConstraint>

Module.Display

brewkedink.wfc

BrewedInk.WFC.Module

Module.Display Field

Every module must have a unique display name. This name is used for hashing and equality checks.

```
public string Display;
```

Field Value

System.String

Module.Weight

brewkedink.wfc

BrewedInk.WFC.Module

Module.Weight Field

A module's weight controls how likely it is to be chosen randomly during a slot collapse. A module with a with a weight of zero will be never be randomly picked.

```
public float Weight;
```

Field Value

System.Single

Slot

brewkedink.wfc

BrewedInk.WFC

Slot Class

In the WFC algorithm, a Slot is a place that holds one Module. As the Wave Function is being collapsed, the slot may have many potential modules available to it.
As the collapse continues, modules are removed as possibilities from the slot, until there is only one module possible for the slot.

```
public class Slot
```

Inheritance System.Object Slot

Fields

Coordinate

The slot's position in the larger generation space.

SlotCannotHaveEmptyModuleSetException

brewkedink.wfc

BrewedInk.WFC

SlotCannotHaveEmptyModuleSetException Class

An exception that is thrown when the WFC tries to remove the last remaining module from a slot's possibility space. A slot cannot have zero possibilities.

```
public class SlotCannotHaveEmptyModuleSetException : System.Exception
```

Inheritance System.Object System.Exception SlotCannotHaveEmptyModuleSetException

SlotCannotSelectUnavailableModule

brewkedink.wfc

BrewedInk.WFC

SlotCannotSelectUnavailableModule Class

An exception that is thrown if a slot tries to select a module that is no longer possible, due to constraints.

```
public class SlotCannotSelectUnavailableModule : System.Exception
```

Inheritance System.Object System.Exception SlotCannotSelectUnavailableModule

SlotEdge

brewkedink.wfc

BrewedInk.WFC

SlotEdge Class

A SlotEdge is the connection between two slots. In a simple 2D 1X2 grid case, there are two slots, and one slot edge connecting them.
The term "edge" comes from Graph Theory. These edges are directional.

```
public class SlotEdge
```

Inheritance System.Object SlotEdge

Fields

Source

The origin slot of the edge.

Target

The destination slot of the edge.

SlotEdge.Source

brewkedink.wfc

BrewedInk.WFC.SlotEdge

SlotEdge.Source Field

The origin slot of the edge.

```
public Slot Source;
```

Field Value

Slot

SlotEdge.Target

brewkedink.wfc

BrewedInk.WFC.SlotEdge

SlotEdge.Target Field

The destination slot of the edge.

```
public Slot Target;
```

Field Value

Slot

SlotModuleRemoved

brewkedink.wfc

BrewedInk.WFC

SlotModuleRemoved Class

A progress instance that explains what module was removed from a slot's possibility space.

```
public class SlotModuleRemoved : BrewedInk.WFC.WFCProgress
```

Inheritance System.Object WFCProgress SlotModuleRemoved

Fields

module

The module that is no longer available for the slot

slot

The slot that has had a module removed.

SlotModuleRemoved.module

brewkedink.wfc

BrewedInk.WFC.SlotModuleRemoved

SlotModuleRemoved.module Field

The module that is no longer available for the slot

```
public Module module;
```

Field Value

Module

SlotModuleRemoved.slot

brewkedink.wfc

BrewedInk.WFC.SlotModuleRemoved

SlotModuleRemoved.slot Field

The slot that has had a module removed.

```
public Slot slot;
```

Field Value

Slot

SlotModuleSelected

brewkedink.wfc

BrewedInk.WFC

SlotModuleSelected Class

A progress instance that explains what module was selected for a slot. When this progress type shows up, it means the superposition for the given slot has collapsed to the given module.

```
public class SlotModuleSelected : BrewedInk.WFC.WFCProgress
```

Inheritance System.Object WFCProgress SlotModuleSelected

Fields

module

The module that has been selected for the given slot

slot

The slot whose superposition has collapsed.

SlotModuleSelected.module

brewkedink.wfc

BrewedInk.WFC.SlotModuleSelected

SlotModuleSelected.module Field

The module that has been selected for the given slot

```
public Module module;
```

Field Value

Module

SlotModuleSelected.slot

brewkedink.wfc

BrewedInk.WFC.SlotModuleSelected

SlotModuleSelected.slot Field

The slot whose superposition has collapsed.

```
public Slot slot;
```

Field Value

Slot

Slot.Coordinate

brewkedink.wfc

BrewedInk.WFC.Slot

Slot.Coordinate Field

The slot's position in the larger generation space.

```
public Vector3Int Coordinate;
```

Field Value

UnityEngine.Vector3Int

SpriteConfig

brewkedink.wfc

SpriteConfig Class

A type of WFCConfigObject that lets you generate sprite modules with socket connective constraints.

```
public class SpriteConfig : BrewedInk.WFC.WCFConfigObject<global::SpriteConfigModuleObject, global::SpriteConfigModule>
```

Inheritance System.Object UnityEngine.Object UnityEngine.ScriptableObject WCFConfigObject BrewedInk.WFC.WCFConfigObject<SpriteConfigModuleObject,SpriteConfigModule> SpriteConfig

WCFConfigObject

brewkedink.wfc

BrewedInk.WFC

WCFConfigObject Class

A ScriptableObject wrapper that streamlines creating GenerationSpace instances from MonoBehaviours.

```
public abstract class WCFConfigObject : UnityEngine.ScriptableObject
```

Inheritance System.Object 🡒 UnityEngine.Object 🡒 UnityEngine.ScriptableObject 🡒 WCFConfigObject

Derived
↳ WCFConfigObject<TModuleObject,TModule>

Methods

Create().md 'BrewedInk.WFC.WCFConfigObject.Create()')

Create a blank GenerationSpace from the configuration.
This is a useful method for creating generation spaces pre-configured with edges, slots, and modules.

TryGetSprite(Module, Sprite).md 'BrewedInk.WFC.WCFConfigObject.TryGetSprite(BrewedInk.WFC.Module, UnityEngine.Sprite)')

If a module has a Sprite associated with it, you can use this method to try and get it. Depending on the type of Module, there may not be any Sprite available.

WCFConfigObject.Create()

brewkedink.wfc

BrewedInk.WFC.WCFConfigObject

WCFConfigObject.Create() Method

Create a blank GenerationSpace from the configuration.
This is a useful method for creating generation spaces pre-configured with edges, slots, and modules.

```
public abstract BrewedInk.WFC.GenerationSpace Create();
```

Returns

GenerationSpace
A new Generation Space

WCFConfigObject.TModuleObject.TModule.

brewkedink.wfc

BrewedInk.WFC

WCFConfigObject<TModuleObject,TModule> Class

```
public abstract class WCFConfigObject<TModuleObject,TModule> : BrewedInk.WFC.WCFConfigObject
    where TModuleObject : BrewedInk.WFC.ModuleObject<TModule>
    where TModule : BrewedInk.WFC.Module
```

Type parameters

`TModuleObject`

`TModule`

Inheritance System.Object 🡒 UnityEngine.Object 🡒 UnityEngine.ScriptableObject 🡒 WCFConfigObject 🡒 WCFConfigObject<TModuleObject,TModule>

Derived
↳ SpriteConfig

Fields

seed

If the useSeed boolean is true, what seed should be used? This property should be ignored if useSeed is false.

useSeed

Should the GenerationSpace instances that this instance creates use a seed?/>

Methods

CreateSpace().md 'BrewedInk.WFC.WCFConfigObject<TModuleObject,TModule>.CreateSpace()')

You need to create a Generation Space given a set of user defined modules and configuration.

TryGetObject(Module, TModuleObject).md 'BrewedInk.WFC.WCFConfigObject<TModuleObject,TModule>.TryGetObject(BrewedInk.WFC.Module, TModuleObject)')

Given the typeless Module, get the wrapper object for it. The wrapper object may have additional metadata about the module that isn't used in the WFC algorithm, but may be useful for other parts of the game.

WCFConfigObject.TModuleObject.TModule..CreateSpace()

brewkedink.wfc

BrewedInk.WFC.WCFConfigObject<TModuleObject,TModule>

WCFConfigObject<TModuleObject,TModule>.CreateSpace() Method

You need to create a Generation Space given a set of user defined modules and configuration.

```
protected abstract BrewedInk.WFC.GenerationSpace CreateSpace();
```

Returns

GenerationSpace
You should return a new GenerationSpace instance that respects the seed properties.

WCFConfigObject.TModuleObject.TModule..seed

brewkedink.wfc

BrewedInk.WFC.WCFConfigObject<TModuleObject,TModule>

WCFConfigObject<TModuleObject,TModule>.seed Field

If the useSeed boolean is true, what seed should be used? This property should be ignored if useSeed is false.

```
public int seed;
```

Field Value

System.Int32

WCFConfigObject.TModuleObject.TModule..TryGetObject(Module.TModuleObject)

brewkedink.wfc

BrewedInk.WFC.WCFConfigObject<TModuleObject,TModule>

WCFConfigObject<TModuleObject,TModule>.TryGetObject(Module, TModuleObject) Method

Given the typeless Module, get the wrapper object for it. The wrapper object may have additional metadata about the module that isn't used in the WFC algorithm, but may be useful for other parts of the game.

```
public bool TryGetObject(BrewedInk.WFC.Module module, out TModuleObject moduleObject);
```

Parameters

`module` Module
The module you want to get the wrapper object for

`moduleObject` TModuleObject
The wrapper object for the Module. Or null if there is no wrapper object for the given Module

Returns

System.Boolean
True if there was an associated wrapper object with the module, or false.

WCFConfigObject.TModuleObject.TModule..useSeed

brewkedink.wfc

BrewedInk.WFC.WCFConfigObject<TModuleObject,TModule>

WCFConfigObject<TModuleObject,TModule>.useSeed Field

Should the GenerationSpace instances that this instance creates use a seed?/>

```
public bool useSeed;
```

Field Value

System.Boolean

WCFConfigObject.TryGetSprite(Module.Sprite)

brewkedink.wfc

BrewedInk.WFC.WCFConfigObject

WCFConfigObject.TryGetSprite(Module, Sprite) Method

If a module has a Sprite associated with it, you can use this method to try and get it. Depending on the type of Module, there may not be any Sprite available.

```
public abstract bool TryGetSprite(BrewedInk.WFC.Module module, out UnityEngine.Sprite sprite);
```

Parameters

`module` Module
The module you'd like to get a sprite for.

`sprite` UnityEngine.Sprite
The Sprite associated with the given module. The sprite will be null if there is no associated Sprite for the module.

Returns

System.Boolean
True if there is any sprite associated with the module, false otherwise. If the method returns false, you can expect the sprite to be null.

WFCError

brewkedink.wfc

BrewedInk.WFC

WFCError Class

A WFCError instance could be returned by the WFC algorithm. If you receive one of these instances, it means something with the process has broken.

```
public class WFCError : BrewedInk.WFC.WFCProgress
```

Inheritance System.Object WFCProgress WFCError

WFCGridRenderer

brewkedink.wfc

BrewedInk.WFC

WFCGridRenderer Class

A wrapper class for displaying a helpful 2D visual for a WFC run.

```
public class WFCGridRenderer : UnityEngine.MonoBehaviour
```

Inheritance System.Object UnityEngine.Object UnityEngine.Component UnityEngine.Behaviour UnityEngine.MonoBehaviour WFCGridRenderer

Fields

board

The sprite to use for the bounds of the board.

config

Any configuration that contains sprites for previews.

WFCGridRenderer.board

brewkedink.wfc

BrewedInk.WFC.WFCGridRenderer

WFCGridRenderer.board Field

The sprite to use for the bounds of the board.

```
public SpriteRenderer board;
```

Field Value

UnityEngine.SpriteRenderer

WFCGridRenderer.config

brewkedink.wfc

BrewedInk.WFC.WFCGridRenderer

WFCGridRenderer.config Field

Any configuration that contains sprites for previews.

```
public WCFConfigObject config;
```

Field Value

WCFConfigObject

WFCProgress

brewkedink.wfc

BrewedInk.WFC

WFCProgress Class

The WFCProgress class represents a unit of progress in the WFC algorithm. As the algorithm is run, a sequence of WCFProgress items will be returned.
Different types of progress will be returned, and you can switch on the type to gain insights to the algorithm's decisions...

```
public class WFCProgress
```

Inheritance System.Object WFCProgress

Derived
↳ SlotModuleRemoved
↳ SlotModuleSelected
↳ WFCError

WFCProgressExtensions

brewkedink.wfc

BrewedInk.WFC

WFCProgressExtensions Class

A Utility class for IEnumerable sets of WFCProgress

```
public static class WFCProgressExtensions
```

Inheritance System.Object WFCProgressExtensions

Methods

RunAsCoroutine(IEnumerable<WFCProgress>, MonoBehaviour, float).md 'BrewedInk.WFC.WFCProgressExtensions.RunAsCoroutine(System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress>, UnityEngine.MonoBehaviour, float)')

A method to run any WFC operation within a Unity Coroutine with frame time budgeting. This method will process as many elements in the WFC operation as possible in on frame, then yield a render frame, and continue the operation on the next frame.
This is the recommended way to run WFC operations in a realtime game.
This returns a WFCProgressObserver object which you can attach progress callbacks and inspect the running state.

RunAsImmediate(IEnumerable<WFCProgress>, Action<WFCProgress>).md 'BrewedInk.WFC.WFCProgressExtensions.RunAsImmediate(System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress>, System.Action<BrewedInk.WFC.WFCProgress>)')

Completely iterate through a WFC operation in one frame. Using this method will likely cause lag in your game, and it is only recommended for testing or advanced use.

WFCProgressExtensions.RunAsCoroutine(IEnumerable.WFCProgress..MonoBehaviour.float)

brewkedink.wfc

BrewedInk.WFC.WFCProgressExtensions

WFCProgressExtensions.RunAsCoroutine(IEnumerable<WFCProgress>, MonoBehaviour, float) Method

A method to run any WFC operation within a Unity Coroutine with frame time budgeting. This method will process as many elements in the WFC operation as possible in on frame, then yield a render frame, and continue the operation on the next frame.
This is the recommended way to run WFC operations in a realtime game.
This returns a WFCProgressObserver object which you can attach progress callbacks and inspect the running state.

```
public static BrewedInk.WFC.WFCProgressObserver RunAsCoroutine(this System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress> operation, UnityEngine.MonoBehaviour context, float timeBudgetPerFrame=
0.1f);
```

Parameters

`operation` System.Collections.Generic.IEnumerable<WFCProgress>
Any WFC operation. This could be the return value from any Collapse related function in the GenerationSpace

`context` UnityEngine.MonoBehaviour
Some Monobehaviour to run the Coroutine in.

`timeBudgetPerFrame` System.Single
How much is available per frame to operate on the WFC sequence? This should be a low number, so that it doesn't cause hitches in your game.

Returns

WFCProgressObserver
a WFCProgressObserver

WFCProgressExtensions.RunAsImmediate(IEnumerable.WFCProgress..Action.WFCProgress.)

brewkedink.wfc

BrewedInk.WFC.WFCProgressExtensions

WFCProgressExtensions.RunAsImmediate(IEnumerable<WFCProgress>, Action<WFCProgress>) Method

Completely iterate through a WFC operation in one frame. Using this method will likely cause lag in your game, and it is only recommended for testing or advanced use.

```
public static void RunAsImmediate(this System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress> operation, System.Action<BrewedInk.WFC.WFCProgress> progressHandler=null);
```

Parameters

`operation` System.Collections.Generic.IEnumerable<WFCProgress>
Any WFC operation. This could be the return value from any Collapse related function in the GenerationSpace

`progressHandler` System.Action<WFCProgress>
An optional callback that runs every time a WFCProgress element is processed.

WFCProgressObserver

brewkedink.wfc

BrewedInk.WFC

WFCProgressObserver Class

A wrapper class around the invocation of a WFC collapse operation.
You should use this in conjunction with RunAsCoroutine(IEnumerable<WFCProgress>, MonoBehaviour, float).md 'BrewedInk.WFC.WFCProgressExtensions.RunAsCoroutine(System.Collections.Generic.IEnumerable<BrewedInk.WFC.WFCProgress>, UnityEngine.MonoBehaviour, float)')

```
public class WFCProgressObserver
```

Inheritance System.Object 🡒 WFCProgressObserver

Properties

IsComplete

Returns true when the operation is complete. This will be true the instant BEFORE the onComplete callbacks are invoked.

ProgressCount

The number of progress events received for this operation. There isn't a knowable upper bound, but you can use this to track that progress is happening.

SlotSelections

A set of selections you can use inspect to see what has finished. If you want an event driven approach, use OnSelectedModule(Action<SlotModuleSelected>).md 'BrewedInk.WFC.WFCProgressObserver.OnSelectedModule(System.Action<BrewedInk.WFC.SlotModuleSelected>)')

Methods

OnCompleted(Action).md 'BrewedInk.WFC.WFCProgressObserver.OnCompleted(System.Action)')

A callback that executes when the WFC operation has completed. This happens right after the WFCProgressObserver.OnComplete is set to true.

OnProgress(Action<WFCProgress>).md 'BrewedInk.WFC.WFCProgressObserver.OnProgress(System.Action<BrewedInk.WFC.WFCProgress>)')

A callback that executes anytime the WFC has new progress. Progress elements can represent a module being removed, or a module being selected, an error, or a general propagation of data.

OnSelectedModule(Action<Slot,Module>).md 'BrewedInk.WFC.WFCProgressObserver.OnSelectedModule(System.Action<BrewedInk.WFC.Slot,BrewedInk.WFC.Module>)')

A callback that executes anytime a slot collapses to a single module possibility. This is a type of WFCProgress, and will execute right after the OnProgress(Action<WFCProgress>).md 'BrewedInk.WFC.WFCProgressObserver.OnProgress(System.Action<BrewedInk.WFC.WFCProgress>)') callbacks

OnSelectedModule(Action<SlotModuleSelected>).md 'BrewedInk.WFC.WFCProgressObserver.OnSelectedModule(System.Action<BrewedInk.WFC.SlotModuleSelected>)')

A callback that executes anytime a slot collapses to a single module possibility. This is a type of WFCProgress, and will execute right after the OnProgress(Action<WFCProgress>).md 'BrewedInk.WFC.WFCProgressObserver.OnProgress(System.Action<BrewedInk.WFC.WFCProgress>)') callbacks

OnSelectedModule<TModule>(Action<Slot,TModule>).md 'BrewedInk.WFC.WFCProgressObserver.OnSelectedModule<TModule>(System.Action<BrewedInk.WFC.Slot,TModule>)')

A callback that executes anytime a slot collapses to a single module possibility. This is a type of WFCProgress, and will execute right after the OnProgress(Action<WFCProgress>).md 'BrewedInk.WFC.WFCProgressObserver.OnProgress(System.Action<BrewedInk.WFC.WFCProgress>)') callbacks

WFCProgressObserver.IsComplete

brewkedink.wfc

BrewedInk.WFC.WFCProgressObserver

WFCProgressObserver.IsComplete Property

Returns true when the operation is complete. This will be true the instant BEFORE the onComplete callbacks are invoked.

```
public bool IsComplete { get; set; }
```

Property Value

System.Boolean

WFCProgressObserver.OnCompleted(Action)

brewkedink.wfc

BrewedInk.WFC.WFCProgressObserver

WFCProgressObserver.OnCompleted(Action) Method

A callback that executes when the WFC operation has completed. This happens right after the WFCProgressObserver.OnComplete is set to true.

```
public BrewedInk.WFC.WFCProgressObserver OnCompleted(System.Action onComplete);
```

Parameters

`onComplete` System.Action
a callback to run on completion

Returns

WFCProgressObserver
The same WFCProgressObserver

WFCProgressObserver.OnProgress(Action.WFCProgress.)

brewkedink.wfc

BrewedInk.WFC.WFCProgressObserver

WFCProgressObserver.OnProgress(Action<WFCProgress>) Method

A callback that executes anytime the WFC has new progress. Progress elements can represent a module being removed, or a module being selected, an error, or a general propagation of data.

```
public BrewedInk.WFC.WFCProgressObserver OnProgress(System.Action<BrewedInk.WFC.WFCProgress> onProgress);
```

Parameters

`onProgress` System.Action<WFCProgress>
A callback to run on progress

Returns

WFCProgressObserver
The same WFCProgressObserver

WFCProgressObserver.OnSelectedModule(Action.SlotModuleSelected.)

brewkedink.wfc

BrewedInk.WFC.WFCProgressObserver

WFCProgressObserver.OnSelectedModule(Action<SlotModuleSelected>) Method

A callback that executes anytime a slot collapses to a single module possibility. This is a type of WFCProgress, and will execute right after the OnProgress(Action<WFCProgress>).md 'BrewedInk.WFC.WFCProgressObserver.OnProgress(System.Action<BrewedInk.WFC.WFCProgress>)') callbacks

```
public BrewedInk.WFC.WFCProgressObserver OnSelectedModule(System.Action<BrewedInk.WFC.SlotModuleSelected> selection);
```

Parameters

`selection` System.Action<SlotModuleSelected>
A callback that takes a general selection. The module isn't typed, so you'll need to type-check it.

Returns

WFCProgressObserver
The same WFCProgressObserver

WFCProgressObserver.OnSelectedModule(Action.Slot.Module.)

brewkedink.wfc

BrewedInk.WFC.WFCProgressObserver

WFCProgressObserver.OnSelectedModule(Action<Slot,Module>) Method

A callback that executes anytime a slot collapses to a single module possibility. This is a type of WFCProgress, and will execute right after the OnProgress(Action<WFCProgress>).md 'BrewedInk.WFC.WFCProgressObserver.OnProgress(System.Action<BrewedInk.WFC.WFCProgress>)') callbacks

```
public BrewedInk.WFC.WFCProgressObserver OnSelectedModule(System.Action<BrewedInk.WFC.Slot,BrewedInk.WFC.Module> selection);
```

Parameters

`selection` System.Action<Slot,Module>
A callback that takes a slot and a general selection. The module isn't typed, so you'll need to type-check it.

Returns

WFCProgressObserver
The same WFCProgressObserver

WFCProgressObserver.OnSelectedModule.TModule.(Action.Slot.TModule.)

brewkedink.wfc

BrewedInk.WFC.WFCProgressObserver

WFCProgressObserver.OnSelectedModule<TModule>(Action<Slot,TModule>) Method

A callback that executes anytime a slot collapses to a single module possibility. This is a type of WFCProgress, and will execute right after the OnProgress(Action<WFCProgress>).md 'BrewedInk.WFC.WFCProgressObserver.OnProgress(System.Action<BrewedInk.WFC.WFCProgress>)') callbacks

```
public BrewedInk.WFC.WFCProgressObserver OnSelectedModule<TModule>(System.Action<BrewedInk.WFC.Slot,TModule> selection);
```

Type parameters

`TModule`
the type of module this callback applies for.

Parameters

`selection` System.Action<Slot,TModule.md#BrewedInk_WFC_WFCProgressObserver_OnSelectedModule_TModule_(System_Action_BrewedInk_WFC_Slot_TModule_)_TModule 'BrewedInk.WFC.WFCProgressObserver.OnSelectedModule<TModule>(System.Action<BrewedInk.WFC.Slot,TModule>).TModule')>
A callback that takes a slot and typed module. This callback only fires when a module of the given type is selected

Returns

WFCProgressObserver
The same WFCProgressObserver

WFCProgressObserver.ProgressCount

brewkedink.wfc

BrewedInk.WFC.WFCProgressObserver

WFCProgressObserver.ProgressCount Property

The number of progress events received for this operation. There isn't a knowable upper bound, but you can use this to track that progress is happening.

```
public int ProgressCount { get; set; }
```

Property Value

System.Int32

WFCProgressObserver.SlotSelections

brewkedink.wfc

BrewedInk.WFC.WFCProgressObserver

WFCProgressObserver.SlotSelections Property

A set of selections you can use inspect to see what has finished. If you want an event driven approach, use OnSelectedModule(Action<SlotModuleSelected>).md 'BrewedInk.WFC.WFCProgressObserver.OnSelectedModule(System.Action<BrewedInk.WFC.SlotModuleSelected>)')

```
public System.Collections.Generic.List<BrewedInk.WFC.SlotModuleSelected> SlotSelections { get; set; }
```

Property Value

System.Collections.Generic.List<SlotModuleSelected>