CMGTGraph

0.2.2

Generated by Doxygen 1.8.18

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

CMGTGraph . . . . . ??
CMGTGraph.Algorithms . . . . . ??
CMGTGraph.Calculators . . . . . ??
CMGTGraph.Logging . . . . . ??
CMGTGraph.Preprocessed . . . . . ??
CMGTGraph.Types . . . . . ??

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

CMGTGraph.Algorithms.Algorithms . . . . . ??
CMGTGraph.Calculators.ICalculator< in in T > . . . . . ??
CMGTGraph.Calculators.ICalculator< Point > . . . . . ??
    CMGTGraph.Calculators.PointCalculator . . . . . ??
    CMGTGraph.Calculators.PointDiagonalCalculator . . . . . ??
    CMGTGraph.Calculators.PointManhattanCalculator . . . . . ??
CMGTGraph.Calculators.ICalculator< PointF > . . . . . ??
    CMGTGraph.Calculators.PointFCalculator . . . . . ??
    CMGTGraph.Calculators.PointFDiagonalCalculator . . . . . ??
    CMGTGraph.Calculators.PointFManhattanCalculator . . . . . ??
IEquatable
    CMGTGraph.Types.Point . . . . . ??
    CMGTGraph.Types.PointF . . . . . ??
IEquatable< Node< T >>
    CMGTGraph.Algorithms.Algorithms.Node< T > . . . . . ??
        CMGTGraph.Algorithms.Algorithms.DijkstraNode< T > . . . . . ??
            CMGTGraph.Algorithms.Algorithms.AStarNode< T > . . . . . ??
CMGTGraph.IReadOnlyGraph< T > . . . . . ??
    CMGTGraph.Graph< T > . . . . . ??
    CMGTGraph.Preprocessed.IAltPreprocessedReadOnlyGraph< T > . . . . . ??
KeyNotFoundException
    CMGTGraph.Graph< T >.NodeNotFoundException . . . . . ??
CMGTGraph.Logging.Logger . . . . . ??
CMGTGraph.Algorithms.Algorithms.PathFindingResult< T > . . . . . ??

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CMGTGraph.Algorithms.Algorithms . . . . . ??

CMGTGraph.Algorithms.Algorithms.AStarNode< T >
A Node specifically built for the purposes of the A∗ algorithm. It saves the current data it represents (the thing from the graph), the predecessor of this node in the current run of the algorithm, the accumulated path length to this node currently, the estimated distance from this node to the finish and an option to get the estimated complete path length going through this node . . . . ??

CMGTGraph.Algorithms.Algorithms.DijkstraNode< T >
A node specifically tailored to the dijkstra algorithm with a predecessor and information about the current path length to this node . . . . . ??

CMGTGraph.Graph< T >
The representation of a graph . . . . . ??

CMGTGraph.Preprocessed.IAltPreprocessedReadOnlyGraph< T >
This interface is very much not finished at all. In fact it's not even started yet. :^) . . . . . ??

CMGTGraph.Calculators.ICalculator< in in T >
An interface for a calculator that can be used to calculate the distance between two objects of type T . . . . . ??

CMGTGraph.IReadOnlyGraph< T > . . . . . ??

CMGTGraph.Logging.Logger . . . . . ??

CMGTGraph.Algorithms.Algorithms.Node< T >
A basic node that links an entry in a node with a predecessor . . . . . ??

CMGTGraph.Graph< T >.NodeNotFoundException
An exception that can be thrown when a specific node is not found is not found in the graph. It contains the node that was requested to be found in string represantation . . . . . ??

CMGTGraph.Algorithms.Algorithms.PathFindingResult< T >
A struct containing data about the path-finding query . . . . . ??

CMGTGraph.Types.Point
A simple class that represents a point in 2D space using integers . . . . . ??

CMGTGraph.Calculators.PointCalculator
A normal calculator to calculate the distance between to integer Points (Point) . . . . . ??

CMGTGraph.Calculators.PointDiagonalCalculator
A special calculator using Diagonal distance on integer points (Point). Use this if you can move in 8 directions . . . . . ??

CMGTGraph.Types.PointF
A simple class that represents a point in 2D space with floating point values . . . . . ??

CMGTGraph.Calculators.PointFCalculator . . . . . ??

CMGTGraph.Calculators.PointFDiagonalCalculator
A special calculator using Diagonal distance on integer points (Point). Use this if you can move in 8 directions . . . . . ??
CMGTGraph.Calculators.PointFManhattanCalculator . . . . . ??
CMGTGraph.Calculators.PointManhattanCalculator . . . . . ??

Chapter 4

Namespace Documentation

4.1 CMGTGraph Namespace Reference

Classes

- class Graph

  The representation of a graph.
- interface IReadOnlyGraph

4.2 CMGTGraph.Algorithms Namespace Reference

Classes

- class Algorithms

4.3 CMGTGraph.Calculators Namespace Reference

Classes

- interface ICalculator

  An interface for a calculator that can be used to calculate the distance between two objects of type T
- class PointCalculator

  A normal calculator to calculate the distance between to integer Points (Point)
- class PointDiagonalCalculator

  A special calculator using Diagonal distance on integer points (Point). Use this if you can move in 8 directions.
- class PointFCalculator
- class PointFDiagonalCalculator

  A special calculator using Diagonal distance on integer points (Point). Use this if you can move in 8 directions.
- class PointFManhattanCalculator
- class PointManhattanCalculator

4.4 CMGTGraph.Logging Namespace Reference

Classes

- class Logger

4.5 CMGTGraph.Preprocessed Namespace Reference

Classes

- interface IAltPreprocessedReadOnlyGraph

  This interface is very much not finished at all. In fact it's not even started yet. :^)

4.6 CMGTGraph.Types Namespace Reference

Classes

- class Point

  A simple class that represents a point in 2D space using integers.
- class PointF

  A simple class that represents a point in 2D space with floating point values.

Chapter 5

Class Documentation

5.1 CMGTGraph.Algorithms.Algorithms Class Reference

Classes

- class AStarNode

  A Node specifically built for the purposes of the A∗ algorithm. It saves the current data it represents (the thing from the graph), the predecessor of this node in the current run of the algorithm, the accumulated path length to this node currently, the estimated distance from this node to the finish and an option to get the estimated complete path length going through this node.
- class DijkstraNode

  A node specifically tailored to the dijkstra algorithm with a predecessor and information about the current path length to this node.
- class Node

  A basic node that links an entry in a node with a predecessor.
- struct PathFindingResult

  A struct containing data about the path-finding query.

Static Public Member Functions

- static List< T > AStarSolve< T > (this IReadOnlyGraph< T > graph, T start, T end, ICalculator< T > calculator=null)

  Get a path between two points in the graph using the A∗ algorithm. If no path can be found, an empty list is returned.
- static PathFindingResult< T > AStarSolveWithInfo< T > (this IReadOnlyGraph< T > g, T start, T end, ICalculator< T > calculator=null)

  Get a path between two points in the graph using the A∗ algorithm. A list of the visited nodes is also returned. If no path can be found, the PathFindingResult<T> will be empty, but no members will be null.
  PathFindingResult<T> will contain the found path, the nodes that were queued to be evaluated (in PathFinding↩ Result<T>.OpenNodes) and the nodes that were finally evaluated (in PathFindingResult<T>.ClosedNodes)
- static List< T > **DijkstraSolve**< **T** > (this IReadOnlyGraph< T > graph, T start, T end)
- static PathFindingResult< T > **DijkstraSolveWithInfo**< **T** > (this IReadOnlyGraph< T > g, T start, T end)
- static List< T > IterativeBfsSolve< T > (this IReadOnlyGraph< T > graph, T start, T end)

  Use the iterative bfs algorithm to find a path between start and end .
- static PathFindingResult< T > IterativeBfsSolveWithInfo< T > (this IReadOnlyGraph< T > graph, T start, T end)

  Use iterative BFS to find a path between start and end . This method will also return the visited nodes in the process. The returned PathFindingResult<T> will contain the path (in PathFindingResult<T>.Path), and the visited nodes (in PathFindingResult<T>.OpenNodes). PathFindingResult<T>.ClosedNodes will be empty as that is not applicable here.
- static List< T > RecursiveSolve< T > (this IReadOnlyGraph< T > graph, T start, T end)

  Kind of the worst kind of pathfinding you can choose, ever.
  It will (eventually) return a path between start and end using a recursive algorithm.

Static Private Member Functions

- static AStarNode< T > AStarGetMostPromisingNode< T > (HashSet< AStarNode< T >> open)

  Get the most promising node from the provided open list. It will search for the node with the lowest F value (AStarNode<T>.EstimatedCompletePathLength)
- static void AStarExpandNode< T > (DijkstraNode< T > node, T finish, IEnumerable< T > neighbors, ICollection< AStarNode< T >> open, ICollection< AStarNode< T >> closed, ICalculator< T > calculator)

  Expand a node, which means add all the neighbors to the open list, initialize them with the correct values (if that hasn't been done) and update them if necessary (a better path to them is found)
- static void DijkstraExpandNode< T > (DijkstraNode< T > node, HashSet< T > neighbors, ICollection< DijkstraNode< T >> open, ICollection< DijkstraNode< T >> closed, ICalculator< T > calculator)

  Expand a node, which basically means to check if any of the neighbors need their values changed (or initialized) and, if they aren't already, the eligible neighbors will be added to the waiting list, waiting to be evaluated.
- static void ThrowOnInvalidInput< T > (this IReadOnlyGraph< T > g, T start, T end)

  Throw if one of the input parameters is not a valid start node.
- static List< T > BuildPath< T > (Node< T > from, IEnumerable< Node< T >> knownNodes)
- static List< T > BuildRecursiveReversePath< T > (Node< T > from, IReadOnlyDictionary< T, Node< T >> knownNodes)

  Recursively build a path from a node using a lookup of values from the node wrapper to the actual containing type.
- static List< T > RecursiveSolve< T > (IReadOnlyGraph< T > graph, T start, T end, List< T > pathTo, int depth=0)

  The actual recursive method that is used to calculate the path.
  This method has an "artificial" recursion anchor at MaxDepth to prevent a StackOverflowException. It will simply stop recursing deeper after a depth higher than MaxDepth.
  Use cautiously :) it can take a while.

Static Private Attributes

- const int **MaxDepth** = 100

5.1.1 Member Function Documentation

5.1.1.1 AStarExpandNode< T >()

```
static void CMGTGraph.Algorithms.Algorithms.AStarExpandNode< T > (
            DijkstraNode< T > node,
            T finish,
            IEnumerable< T > neighbors,
            ICollection< AStarNode< T >> open,
            ICollection< AStarNode< T >> closed,
            ICalculator< T > calculator )  [static], [private]
```

Expand a node, which means add all the neighbors to the open list, initialize them with the correct values (if that hasn't been done) and update them if necessary (a better path to them is found)

Parameters

node	The node to expand
finish	The finish we strive for
neighbors	The neighbors of the passed node (to prevent passing the graph)
open	The list of nodes that might be expanded later
closed	The list of nodes we don't want to expand later
calculator	The calculator we want to use to calculate distances between nodes

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.2 AStarGetMostPromisingNode< T >()

```
static AStarNode<T> CMGTGraph.Algorithms.Algorithms.AStarGetMostPromisingNode< T > (
            HashSet< AStarNode< T >> open ) [static], [private]
```

Get the most promising node from the provided open list. It will search for the node with the lowest F value (AStarNode<T>.EstimatedCompletePathLength)

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.3 AStarSolve< T >()

```
static List<T> CMGTGraph.Algorithms.Algorithms.AStarSolve< T > (
            this IReadOnlyGraph< T > graph,
            T start,
            T end,
            ICalculator< T > calculator = null ) [static]
```

Get a path between two points in the graph using the A∗ algorithm. If no path can be found, an empty list is returned.

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.4 AStarSolveWithInfo< T >()

```
static PathFindingResult<T> CMGTGraph.Algorithms.Algorithms.AStarSolveWithInfo< T > (
            this IReadOnlyGraph< T > g,
            T start,
            T end,
            ICalculator< T > calculator = null ) [static]
```

Get a path between two points in the graph using the A∗ algorithm. A list of the visited nodes is also returned. If no path can be found, the PathFindingResult<T> will be empty, but no members will be null.
PathFindingResult<T> will contain the found path, the nodes that were queued to be evaluated (in PathFinding↩ Result<T>.OpenNodes) and the nodes that were finally evaluated (in PathFindingResult<T>.ClosedNodes)

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.5 BuildPath< T >()

```
static List<T> CMGTGraph.Algorithms.Algorithms.BuildPath< T > (
            Node< T > from,
            IEnumerable< Node< T >> knownNodes )  [static], [private]
```

Build a path from a node (usually the finish node of the pathfinding request) and a collection of known nodes (which will be used as a look up into the graph).

from will be the last node in the built path, as this is the node to get the predecessor from.

Parameters

from	The node to build the recursively build the path from.
knownNodes	

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.6 BuildRecursiveReversePath< T >()

```
static List<T> CMGTGraph.Algorithms.Algorithms.BuildRecursiveReversePath< T > (
            Node< T > from,
            IReadOnlyDictionary< T, Node< T >> knownNodes )  [static], [private]
```

Recursively build a path from a node using a lookup of values from the node wrapper to the actual containing type.

Parameters

from	The node to build from
knownNodes	A look up from the node wrapper to the graph type

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.7 DijkstraExpandNode< T >()

```
static void CMGTGraph.Algorithms.Algorithms.DijkstraExpandNode< T > (
            DijkstraNode< T > node,
            HashSet< T > neighbors,
            ICollection< DijkstraNode< T >> open,
            ICollection< DijkstraNode< T >> closed,
            ICalculator< T > calculator )  [static], [private]
```

Expand a node, which basically means to check if any of the neighbors need their values changed (or initialized) and, if they aren't already, the eligible neighbors will be added to the waiting list, waiting to be evaluated.

Parameters

node	The node to check the neighbors of
neighbors	The neighbors of this node
open	The list of nodes that are already queued for evaluation
closed	The list of nodes that is finished with this shit
calculator	The calculator to use for distance calculation

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.8 IterativeBfsSolve< T >()

```
static List<T> CMGTGraph.Algorithms.Algorithms.IterativeBfsSolve< T > (
            this IReadOnlyGraph< T > graph,
            T start,
            T end )  [static]
```

Use the iterative bfs algorithm to find a path between *start* and *end* .

Parameters

graph	The graph to solve on
start	The start node
end	The end node

Exceptions

Graph<*T*>*.NodeNotFoundException*	This will be thrown if either start or end are in the graph.
InvalidOperationException	This will be thrown when one of the nodes is not passable at the time of asking (politely of course)

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.9 IterativeBfsSolveWithInfo< T >()

```
static PathFindingResult<T> CMGTGraph.Algorithms.Algorithms.IterativeBfsSolveWithInfo< T > (
            this IReadOnlyGraph< T > graph,
            T start,
            T end )  [static]
```

Use iterative BFS to find a path between *start* and *end* . This method will also return the visited nodes in the process.
The returned PathFindingResult<T> will contain the path (in PathFindingResult<T>.Path), and the visited nodes (in PathFindingResult<T>.OpenNodes). PathFindingResult<T>.ClosedNodes will be empty as that is not applicable here.

Parameters

graph	The graph to operate on
start	The node to start searching from
end	The node that is the finish

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.10 RecursiveSolve< T >() [1/2]

```
static List<T> CMGTGraph.Algorithms.Algorithms.RecursiveSolve< T > (
            IReadOnlyGraph< T > graph,
            T start,
            T end,
            List< T > pathTo,
            int depth = 0 )  [static], [private]
```

The actual recursive method that is used to calculate the path.
This method has an "artificial" recursion anchor at MaxDepth to prevent a StackOverflowException. It will simply stop recursing deeper after a depth higher than MaxDepth.
Use cautiously :) it can take a while.

Parameters

graph	The graph to perform the operation on.
start	The start node of the problem
end	The end node.
pathTo	The current path to the start node.
depth	The current depth we are at.

Template Parameters

T	

Returns

Returns the reverse path from start to end

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.11 RecursiveSolve< T >() [2/2]

```
static List<T> CMGTGraph.Algorithms.Algorithms.RecursiveSolve< T > (
            this IReadOnlyGraph< T > graph,
            T start,
            T end )  [static]
```

Kind of the worst kind of pathfinding you can choose, ever.
It will (eventually) return a path between start and end using a recursive algorithm.

Type Constraints

***T* : *IEquatable*<*T*>**

5.1.1.12 ThrowOnInvalidInput< T >()

```
static void CMGTGraph.Algorithms.Algorithms.ThrowOnInvalidInput< T > (
            this IReadOnlyGraph< T > g,
            T start,
            T end )  [static], [private]
```

Throw if one of the input parameters is not a valid start node.

Template Parameters

T	

Exceptions

Graph<*T*>*.NodeNotFoundException*	
InvalidOperationException	

Type Constraints

***T* : *IEquatable*<*T*>**

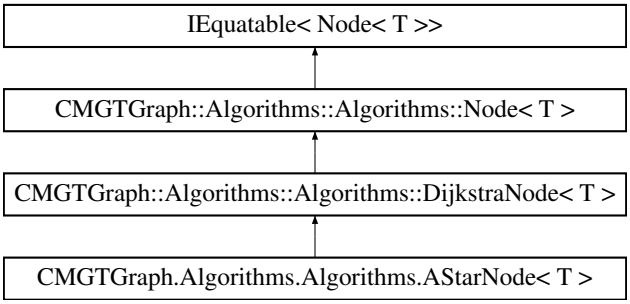
The documentation for this class was generated from the following files:

- CMGTGraph/Algorithms/Algorithms-AStar.cs
- CMGTGraph/Algorithms/Algorithms-RecursivePath.cs
- CMGTGraph/Algorithms/Algorithms-Dijkstra.cs
- CMGTGraph/Algorithms/Algorithms-Helpers.cs
- CMGTGraph/Algorithms/Algorithms-IterativeBFS.cs

5.2 CMGTGraph.Algorithms.Algorithms.AStarNode< T > Class Template Reference

A Node specifically built for the purposes of the A* algorithm. It saves the current data it represents (the thing from the graph), the predecessor of this node in the current run of the algorithm, the accumulated path length to this node currently, the estimated distance from this node to the finish and an option to get the estimated complete path length going through this node.

Inheritance diagram for CMGTGraph.Algorithms.Algorithms.AStarNode< T >:



Public Member Functions

- AStarNode (T data, T predecessor=default, float currentPathLength=0f, float distanceToFinish=float.Max↩ Value)

  Create a new AStarNode. You don't need to specify any values apart from the actual data. They can all be set later. predecessor will by default initialize to

Public Attributes

- float DistanceToFinish

  The estimated distance of this node to the finish. In formal definitions this is often called H or h(x) standing for heuristic.
- float EstimatedCompletePathLength => CurrentPathLength + DistanceToFinish

  The estimated complete path length that would go through this node. It is the length to this node (DijkstraNode<T>.CurrentPathLength) plus the estimated distance to the finish. In formal definitions, this is often called F or f(x) and defined as f(x) = g(x) + h(x)

Additional Inherited Members

5.2.1 Detailed Description

A Node specifically built for the purposes of the A* algorithm. It saves the current data it represents (the thing from the graph), the predecessor of this node in the current run of the algorithm, the accumulated path length to this node currently, the estimated distance from this node to the finish and an option to get the estimated complete path length going through this node.

Template Parameters

T	

Type Constraints

***T* : *IEquatable*<*T*>**

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AStarNode()

```
CMGTGraph.Algorithms.Algorithms.AStarNode< T >.AStarNode (
            T data,
            T predecessor = default,
            float currentPathLength = 0f,
            float distanceToFinish = float.MaxValue )
```

Create a new AStarNode. You don't need to specify any values apart from the actual data. They can all be set later. *predecessor* will by default initialize to

`default`, *currentPathLength* will default to zero and *distanceToFinish* will default to float.MaxValue.

5.2.3 Member Data Documentation

5.2.3.1 DistanceToFinish

```
float CMGTGraph.Algorithms.Algorithms.AStarNode< T >.DistanceToFinish
```

The estimated distance of this node to the finish. In formal definitions this is often called H or h(x) standing for heuristic.

5.2.3.2 EstimatedCompletePathLength

```
float CMGTGraph.Algorithms.Algorithms.AStarNode< T >.EstimatedCompletePathLength => CurrentPathLength
+ DistanceToFinish
```

The estimated complete path length that would go through this node. It is the length to this node (DijkstraNode<T>.CurrentPathLength) plus the estimated distance to the finish. In formal definitions, this is often called F or f(x) and defined as f(x) = g(x) + h(x)

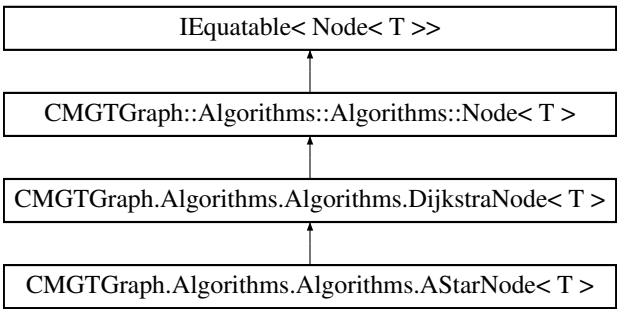
The documentation for this class was generated from the following file:

- CMGTGraph/Algorithms/Algorithms-AStar.cs

5.3 CMGTGraph.Algorithms.Algorithms.DijkstraNode< T > Class Template Reference

A node specifically tailored to the dijkstra algorithm with a predecessor and information about the current path length to this node.

Inheritance diagram for CMGTGraph.Algorithms.Algorithms.DijkstraNode< T >:



Public Member Functions

- DijkstraNode (T data, T predecessor=default, float currentPathLength=float.MaxValue)

Public Attributes

- float CurrentPathLength

  The length of the path to this node.

Additional Inherited Members

5.3.1 Detailed Description

A node specifically tailored to the dijkstra algorithm with a predecessor and information about the current path length to this node.

Type Constraints

***T* : *IEquatable*<*T*>**

5.3.2 Member Data Documentation

5.3.2.1 CurrentPathLength

```
float CMGTGraph.Algorithms.Algorithms.DijkstraNode< T >.CurrentPathLength
```

The length of the path to this node.

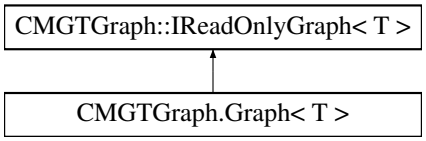
The documentation for this class was generated from the following file:

- CMGTGraph/Algorithms/Algorithms-Dijkstra.cs

5.4 CMGTGraph.Graph< T > Class Template Reference

The representation of a graph.

Inheritance diagram for CMGTGraph.Graph< T >:



Classes

- class NodeNotFoundException

  An exception that can be thrown when a specific node is not found is not found in the graph. It contains the node that was requested to be found in string represantation.

Public Member Functions

- Graph (ICalculator< T > calculator)

  Create a new graph.
- void Add (T node)

  Add a node to the graph. If the node is already in the graph, nothing happens.
- void AddConnection (T nodeA, T nodeB)

  Add a connection between two nodes. If one or both of the nodes don't/doesn't exist in the graph, it/they will be added to the graph. If you pass the same node twice, it will be added to the graph but no connections will be added as to not create loops.
- bool RemoveNode (T node)

  Remove the provided node and all connections to that node.
- bool RemoveConnection (T nodeA, T nodeB)

  Remove the connection between the two nodes.
- HashSet< T > GetPassableConnections (T node)

  Get all the nodes that are connected to this node, provided they are passable.
- bool HaveConnection (T nodeA, T nodeB)

  Test if the provided nodes have a connection in the graph. Returns true if yes.
- void Clear ()

  Delete all nodes and connections from the graph.
- override string ToString ()
- bool Contains (T value)

  Is a specific node in the Graph?
- bool NodeIsPassable (T node)

  Check if a node is passable.
- void MakeImpassable (T node)

  Make a node impassable. Has no effect, if the node is already impassable.
- void MakePassable (T node)

  Make a node passable. Has no effect, if the node is already passable.
- void TogglePassable (T node)

  Toggle, if a node is passable or not, i.e. make it passable when it's not and make it impassable when it is.
- bool IsConnected ()

  Check if the graph is connected (every node has a path to every other node). This takes into account, if the node is passable.

Public Attributes

- int NodeCount => _connections.Count
- HashSet< T > Nodes => new HashSet<T>(_connections.Keys)

Properties

- ICalculator< T > Calculator `[get, set]`

Private Attributes

- readonly Dictionary< T, HashSet< T > > _connections

  The underlying representation of the graph as an adjacency list.
- readonly HashSet< T > _impassable

  The collection of nodes that is not passable.

5.4.1 Detailed Description

The representation of a graph.

Template Parameters

T	T needs to have a robust GetHashCode() implementation, as many operations rely on it in here

Type Constraints

***T* : *IEquatable*<*T*>**

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Graph()

```
CMGTGraph.Graph< T >.Graph (
            ICalculator< T > calculator )
```

Create a new graph.

Parameters

calculator	The calculator that can be used to calculate useful things for the specified type

5.4.3 Member Function Documentation

5.4.3.1 Add()

```
void CMGTGraph.Graph< T >.Add (
            T node )
```

Add a node to the graph. If the node is already in the graph, nothing happens.

5.4.3.2 AddConnection()

```
void CMGTGraph.Graph< T >.AddConnection (
            T nodeA,
            T nodeB )
```

Add a connection between two nodes. If one or both of the nodes don't/doesn't exist in the graph, it/they will be added to the graph. If you pass the same node twice, it will be added to the graph but no connections will be added as to not create loops.

5.4.3.3 Clear()

```
void CMGTGraph.Graph< T >.Clear ( )
```

Delete all nodes and connections from the graph.

5.4.3.4 Contains()

```
bool CMGTGraph.Graph< T >.Contains (
            T value )
```

Is a specific node in the Graph?

Parameters

value	The node to check

Returns

Returns true, if the passed node is in the graph

Implements CMGTGraph.IReadOnlyGraph< T >.

5.4.3.5 HaveConnection()

```
bool CMGTGraph.Graph< T >.HaveConnection (
            T nodeA,
            T nodeB )
```

Test if the provided nodes have a connection in the graph. Returns true if yes.

Exceptions

NodeNotFoundException	Thrown when one of the provided nodes is not in the graph

Implements CMGTGraph.IReadOnlyGraph< T >.

5.4.3.6 MakeImpassable()

```
void CMGTGraph.Graph< T >.MakeImpassable (
            T node )
```

Make a node impassable. Has no effect, if the node is already impassable.

5.4.3.7 MakePassable()

```
void CMGTGraph.Graph< T >.MakePassable (
            T node )
```

Make a node passable. Has no effect, if the node is already passable.

5.4.3.8 RemoveConnection()

```
bool CMGTGraph.Graph< T >.RemoveConnection (
            T nodeA,
            T nodeB )
```

Remove the connection between the two nodes.

Returns

Returns false, if one of the nodes isn't in the graph or if no connection has been removed.

5.4.3.9 RemoveNode()

```
bool CMGTGraph.Graph< T >.RemoveNode (
            T node )
```

Remove the provided node and all connections to that node.

Returns

Returns true, if the node could be removed successfully.

5.4.3.10 TogglePassable()

```
void CMGTGraph.Graph< T >.TogglePassable (
            T node )
```

Toggle, if a node is passable or not, i.e. make it passable when it's not and make it impassable when it is.

5.4.4 Member Data Documentation

5.4.4.1 _connections

```
readonly Dictionary<T, HashSet<T> > CMGTGraph.Graph< T >._connections  [private]
```

The underlying representation of the graph as an adjacency list.

5.4.4.2 _impassable

```
readonly HashSet<T> CMGTGraph.Graph< T >._impassable  [private]
```

The collection of nodes that is not passable.

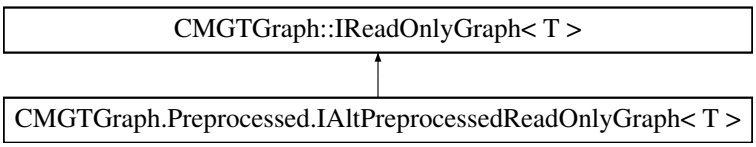
The documentation for this class was generated from the following file:

- CMGTGraph/Graph.cs

5.5 CMGTGraph.Preprocessed.IAltPreprocessedReadOnlyGraph< T > Interface Template Reference

This interface is very much not finished at all. In fact it's not even started yet. :^)

Inheritance diagram for CMGTGraph.Preprocessed.IAltPreprocessedReadOnlyGraph< T >:



Additional Inherited Members

5.5.1 Detailed Description

This interface is very much not finished at all. In fact it's not even started yet. :^)

Template Parameters

T	

Type Constraints

***T* : *IEquatable*<*T*>**

The documentation for this interface was generated from the following file:

- CMGTGraph/Preprocessed/IAltReadOnlyGraph.cs

5.6 CMGTGraph.Calculators.ICalculator< in in T > Interface Template Reference

An interface for a calculator that can be used to calculate the distance between two objects of type T

Public Member Functions

- float SqrDistance (T a, T b)

  Calculate the square distance between the two provided elements.
- float Distance (T a, T b)

  Calculate the distance between the two provided points

5.6.1 Detailed Description

An interface for a calculator that can be used to calculate the distance between two objects of type T

Template Parameters

T	

5.6.2 Member Function Documentation

5.6.2.1 Distance()

```
float CMGTGraph.Calculators.ICalculator< in in T >.Distance (
            T a,
            T b )
```

Calculate the distance between the two provided points

5.6.2.2 SqrDistance()

```
float CMGTGraph.Calculators.ICalculator< in in T >.SqrDistance (
            T a,
            T b )
```

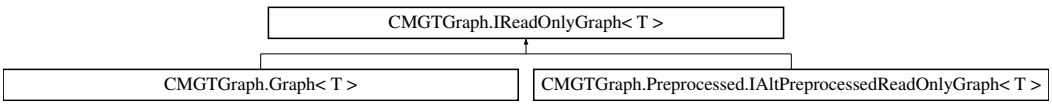
Calculate the square distance between the two provided elements.

The documentation for this interface was generated from the following file:

- CMGTGraph/Calculators/ICalculator.cs

5.7 CMGTGraph.IReadOnlyGraph< T > Interface Template Reference

Inheritance diagram for CMGTGraph.IReadOnlyGraph< T >:



Public Member Functions

- HashSet< T > GetPassableConnections (T node)

  Get all the nodes that are connected to this node, provided they are passable.
- bool HaveConnection (T nodeA, T nodeB)

  Check if two nodes are connected
- bool Contains (T value)

  Check if a node is in the graph.
- bool NodeIsPassable (T node)

  Check if a node is passable.
- bool IsConnected ()

  Check if the graph is connected (every node has a path to every other node). This takes into account, if the node is passable.

Properties

- int NodeCount [get]

  The number of nodes in the graph.
- HashSet< T > Nodes [get]

  Get all the nodes in the graph.
- ICalculator< T > Calculator [get]

  The calculator that can be used to calculate things like SqrDistance between nodes.

5.7.1 Member Function Documentation

5.7.1.1 Contains()

```
bool CMGTGraph.IReadOnlyGraph< T >.Contains (
            T value )
```

Check if a node is in the graph.

Implemented in CMGTGraph.Graph< T >.

5.7.1.2 GetPassableConnections()

```
HashSet<T> CMGTGraph.IReadOnlyGraph< T >.GetPassableConnections (
            T node )
```

Get all the nodes that are connected to this node, provided they are passable.

Parameters

node	

Returns

Implemented in CMGTGraph.Graph< T >.

5.7.1.3 HaveConnection()

```
bool CMGTGraph.IReadOnlyGraph< T >.HaveConnection (
            T nodeA,
            T nodeB )
```

Check if two nodes are connected

Implemented in CMGTGraph.Graph< T >.

5.7.1.4 IsConnected()

```
bool CMGTGraph.IReadOnlyGraph< T >.IsConnected ( )
```

Check if the graph is connected (every node has a path to every other node). This takes into account, if the node is passable.

Returns

Implemented in CMGTGraph.Graph< T >.

5.7.1.5 NodeIsPassable()

```
bool CMGTGraph.IReadOnlyGraph< T >.NodeIsPassable (
            T node )
```

Check if a node is passable.

Parameters

node	

Returns

Implemented in CMGTGraph.Graph< T >.

5.7.2 Property Documentation

5.7.2.1 Calculator

```
ICalculator<T> CMGTGraph.IReadOnlyGraph< T >.Calculator  [get]
```

The calculator that can be used to calculate things like SqrDistance between nodes.

5.7.2.2 NodeCount

```
int CMGTGraph.IReadOnlyGraph< T >.NodeCount  [get]
```

The number of nodes in the graph.

5.7.2.3 Nodes

```
HashSet<T> CMGTGraph.IReadOnlyGraph< T >.Nodes  [get]
```

Get all the nodes in the graph.

The documentation for this interface was generated from the following file:

- CMGTGraph/IReadOnlyGraph.cs

5.8 CMGTGraph.Logging.Logger Class Reference

Public Types

- enum LogLevel {
  LogLevel.None, LogLevel.Error, LogLevel.Warning, LogLevel.Verbose,
  LogLevel.Spam }

  The log level of the logger.

Static Public Member Functions

- static void ResetWriter ()

  Reset the text writer to point to the one it was when changing the text writer through Writer. If the text writer hasn't been changed, this function does nothing.
- static void Spam (string text)

  Spam something to the console in white with the prefix
- static void Log (string text)

  Log something to the console in dark blue with the prefix
- static void Warn (string text)

  Log something to the console in yellow with the prefix
- static void Error (string text)

  Log something to the console in red with the prefix
- static void ColorLog (string text, ConsoleColor color=ConsoleColor.White)

  Log something in the specified console color to the current output text writer. The default color is white.

Static Public Attributes

- static LogLevel LoggingLevel = LogLevel.Error

  The current logging level, the logger is operating at.

Properties

- static TextWriter Writer `[get, set]`

  When you get it, you get the current text writer of this logger.

Static Private Attributes

- static TextWriter _defaultOut

  The default text writer that is saved when the output text writer is changed (e.g. through Writer) to be able to reset it later.

5.8.1 Member Enumeration Documentation

5.8.1.1 LogLevel

```
enum CMGTGraph.Logging.Logger.LogLevel  [strong]
```

The log level of the logger.

Enumerator

None	Print nothing at all.
Error	Print only errors. (Logger.Error)
Warning	Print errors (Logger.Error) and warnings (Logger.Warn)
Verbose	Print errors (Logger.Error), warnings (Logger.Warn) and logs (Logger.Log).
Spam	Print everything from every function in this static class.

5.8.2 Member Function Documentation

5.8.2.1 ColorLog()

```
static void CMGTGraph.Logging.Logger.ColorLog (
            string text,
            ConsoleColor color = ConsoleColor.White )  [static]
```

Log something in the specified console color to the current output text writer. The default color is white.

Parameters

text	The text to log
color	The color to log in

5.8.2.2 Error()

```
static void CMGTGraph.Logging.Logger.Error (
            string text )  [static]
```

Log something to the console in red with the prefix

`CMGTGraph [Error]:  .`

Parameters

text	The text to log as an error

5.8.2.3 Log()

```
static void CMGTGraph.Logging.Logger.Log (
            string text )  [static]
```

Log something to the console in dark blue with the prefix

`CMGTGraph:  .`

Parameters

text	The text to log

5.8.2.4 ResetWriter()

```
static void CMGTGraph.Logging.Logger.ResetWriter ( )  [static]
```

Reset the text writer to point to the one it was when changing the text writer through Writer. If the text writer hasn't been changed, this function does nothing.

5.8.2.5 Spam()

```
static void CMGTGraph.Logging.Logger.Spam (
            string text )  [static]
```

Spam something to the console in white with the prefix

`Spam:  .`

Parameters

text	The text to spam

5.8.2.6 Warn()

```
static void CMGTGraph.Logging.Logger.Warn (
            string text )  [static]
```

Log something to the console in yellow with the prefix

`CMGTGraph [Warning]: .`

Parameters

text	The text to log as a warning

5.8.3 Member Data Documentation

5.8.3.1 _defaultOut

```
TextWriter CMGTGraph.Logging.Logger._defaultOut  [static], [private]
```

The default text writer that is saved when the output text writer is changed (e.g. through Writer) to be able to reset it later.

5.8.3.2 LoggingLevel

```
LogLevel CMGTGraph.Logging.Logger.LoggingLevel = LogLevel.Error  [static]
```

The current logging level, the logger is operating at.

5.8.4 Property Documentation

5.8.4.1 Writer

```
TextWriter CMGTGraph.Logging.Logger.Writer  [static], [get], [set]
```

When you get it, you get the current text writer of this logger.

When you set it, the default text writer will be saved in _defaultOut (to be able to reset it) and the out will be set. If the provided value is null, nothing happens.

In the current implementation, Writer is always equal to Console.Out, as it simply sets the out of the console.

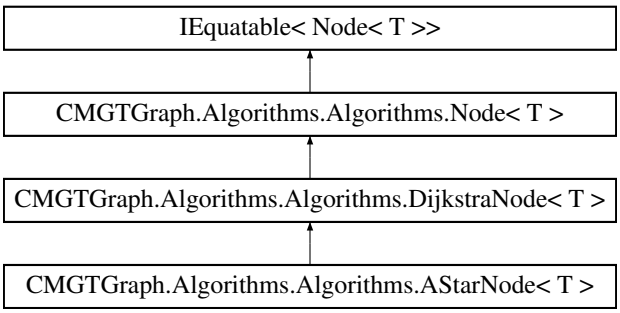
The documentation for this class was generated from the following file:

- CMGTGraph/Logging/Logger.cs

5.9 CMGTGraph.Algorithms.Algorithms.Node< T > Class Template Reference

A basic node that links an entry in a node with a predecessor.

Inheritance diagram for CMGTGraph.Algorithms.Algorithms.Node< T >:



Public Member Functions

- **Node** (T data, T predecessor=default)
- bool Equals (Node< T > other)
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static bool **operator==** (Node< T > left, Node< T > right)
- static bool **operator!=** (Node< T > left, Node< T > right)

Public Attributes

- readonly T Data

  The actual data that is also (hopefully) in the graph
- T Predecessor

  The predecessor to this node

5.9.1 Detailed Description

A basic node that links an entry in a node with a predecessor.

Type Constraints

T **:** ***IEquatable*<*T*>**

5.9.2 Member Data Documentation

5.9.2.1 Data

```
readonly T CMGTGraph.Algorithms.Algorithms.Node< T >.Data
```

The actual data that is also (hopefully) in the graph

5.9.2.2 Predecessor

```
T CMGTGraph.Algorithms.Algorithms.Node< T >.Predecessor
```

The predecessor to this node

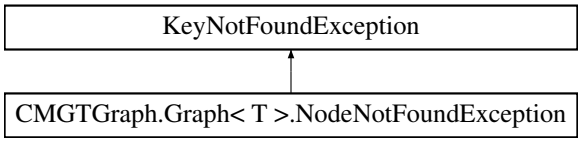
The documentation for this class was generated from the following file:

- CMGTGraph/Algorithms/Algorithms-Helpers.cs

5.10 CMGTGraph.Graph< T >.NodeNotFoundException Class Reference

An exception that can be thrown when a specific node is not found is not found in the graph.
It contains the node that was requested to be found in string represantation.

Inheritance diagram for CMGTGraph.Graph< T >.NodeNotFoundException:



Public Member Functions

- NodeNotFoundException (T value)

  Create a new NodeNotFoundException with the value that was tried to be found, but was not found. This node will be stored in the exception data in string representation.

5.10.1 Detailed Description

An exception that can be thrown when a specific node is not found is not found in the graph.
It contains the node that was requested to be found in string represantation.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 NodeNotFoundException()

```
CMGTGraph.Graph< T >.NodeNotFoundException.NodeNotFoundException (
            T value )
```

Create a new NodeNotFoundException with the value that was tried to be found, but was not found. This node will be stored in the exception data in string representation.

Parameters

value	

The documentation for this class was generated from the following file:

- CMGTGraph/Graph.cs

5.11 CMGTGraph.Algorithms.Algorithms.PathFindingResult< T > Struct Template Reference

A struct containing data about the path-finding query.

Public Member Functions

- **PathFindingResult** (List< T > path, HashSet< T > openNodes, HashSet< T > closedNodes)

Public Attributes

- readonly List< T > **Path**
- readonly HashSet< T > **OpenNodes**
- readonly HashSet< T > **ClosedNodes**

Static Public Attributes

- static PathFindingResult< T > **Empty**

5.11.1 Detailed Description

A struct containing data about the path-finding query.

Template Parameters

T	

Type Constraints

***T* : *IEquatable*<*T*>**

5.11.2 Member Data Documentation

5.11.2.1 Empty

```
PathFindingResult<T> CMGTGraph.Algorithms.Algorithms.PathFindingResult< T >.Empty  [static]
```

Initial value:

```
=>
                new PathFindingResult<T>(new List<T>(), new HashSet<T>(), new HashSet<T>())
```

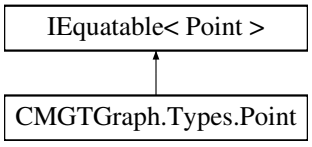
The documentation for this struct was generated from the following file:

- CMGTGraph/Algorithms/Algorithms-Helpers.cs

5.12 CMGTGraph.Types.Point Class Reference

A simple class that represents a point in 2D space using integers.

Inheritance diagram for CMGTGraph.Types.Point:



Public Member Functions

- Point (int x, int y)

  Create a new integer Point. All the values are readonly.
- bool Equals (Point other)
- override bool Equals (object obj)
- override int GetHashCode ()
- override string ToString ()

Static Public Member Functions

- static bool **operator==** (Point left, Point right)
- static bool **operator!=** (Point left, Point right)

Public Attributes

- readonly int **X**
- readonly int **Y**

5.12.1 Detailed Description

A simple class that represents a point in 2D space using integers.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Point()

```
CMGTGraph.Types.Point.Point (
            int x,
            int y )
```

Create a new integer Point. All the values are readonly.

Parameters

x	
y	

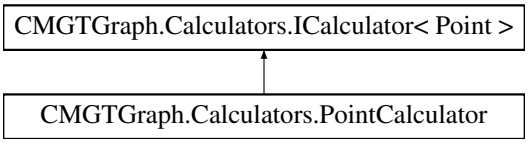
The documentation for this class was generated from the following file:

- CMGTGraph/Types/Point.cs

5.13 CMGTGraph.Calculators.PointCalculator Class Reference

A normal calculator to calculate the distance between to integer Points (Point)

Inheritance diagram for CMGTGraph.Calculators.PointCalculator:



Public Member Functions

- float SqrDistance (Point a, Point b)
- float Distance (Point a, Point b)

Static Public Attributes

- static readonly PointCalculator This

  The static instance of this calculator that should always be null.

Private Member Functions

- PointCalculator ()

  Hidden so you can't construct it on your own. Also does nothing.

Static Private Member Functions

- static void ThrowIfNull (Point a, Point b)

  Throw ArgumentNullException if one of the arguments is null.

5.13.1 Detailed Description

A normal calculator to calculate the distance between to integer Points (Point)

5.13.2 Constructor & Destructor Documentation

5.13.2.1 PointCalculator()

```
CMGTGraph.Calculators.PointCalculator.PointCalculator ( )  [private]
```

Hidden so you can't construct it on your own. Also does nothing.

5.13.3 Member Function Documentation

5.13.3.1 Distance()

```
float CMGTGraph.Calculators.PointCalculator.Distance (
            Point a,
            Point b )
```

Exceptions

NullReferenceException	if one of the arguments is null

5.13.3.2 SqrDistance()

```
float CMGTGraph.Calculators.PointCalculator.SqrDistance (
            Point a,
            Point b )
```

Exceptions

NullReferenceException	if one of the arguments is null

5.13.3.3 ThrowIfNull()

```
static void CMGTGraph.Calculators.PointCalculator.ThrowIfNull (
            Point a,
            Point b )  [static], [private]
```

Throw ArgumentNullException if one of the arguments is null.

5.13.4 Member Data Documentation

5.13.4.1 This

```
readonly PointCalculator CMGTGraph.Calculators.PointCalculator.This  [static]
```

The static instance of this calculator that should always be null.

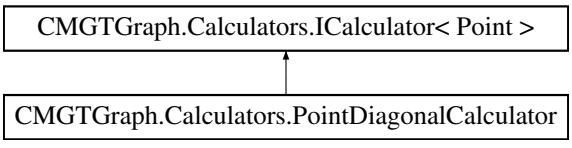
The documentation for this class was generated from the following file:

- CMGTGraph/Calculators/PointCalculator.cs

5.14 CMGTGraph.Calculators.PointDiagonalCalculator Class Reference

A special calculator using Diagonal distance on integer points (Point). Use this if you can move in 8 directions.

Inheritance diagram for CMGTGraph.Calculators.PointDiagonalCalculator:



Public Member Functions

- float SqrDistance (Point a, Point b)
- float Distance (Point a, Point b)

Static Public Attributes

- static readonly PointDiagonalCalculator Octile
- static readonly PointDiagonalCalculator Chebyshev

Private Member Functions

- PointDiagonalCalculator (float straight=1f, float diagonal=1.41421356237f)

Private Attributes

- readonly float _straight

  The value that is used to weigh straight edges.
- readonly float _diagonal

  The value that is used to weigh diagonal edges.

5.14.1 Detailed Description

A special calculator using Diagonal distance on integer points (Point). Use this if you can move in 8 directions.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 PointDiagonalCalculator()

```
CMGTGraph.Calculators.PointDiagonalCalculator.PointDiagonalCalculator (
            float straight = 1f,
            float diagonal = 1.41421356237f )  [private]
```

Create a new calculator with new weights for straight and diagonal edges. The default values are 1 for straight edges and SQRT(2) for diagonal ones.

This implicitly hides the constructor with default values, to prevent creating a new instance on the outside.

5.14.3 Member Data Documentation

5.14.3.1 _diagonal

```
readonly float CMGTGraph.Calculators.PointDiagonalCalculator._diagonal [private]
```

The value that is used to weigh diagonal edges.

5.14.3.2 _straight

```
readonly float CMGTGraph.Calculators.PointDiagonalCalculator._straight [private]
```

The value that is used to weigh straight edges.

5.14.3.3 Chebyshev

```
readonly PointDiagonalCalculator CMGTGraph.Calculators.PointDiagonalCalculator.Chebyshev [static]
```

A calculator using chebyshev diagonal distance, which basically means diagonal and straight edges have the same weight, thus diagonal edges are preferred using this calculator as a heuristic, compared to using octile distance (with Octile).

Read more on grid heuristics here: http://theory.stanford.edu/~amitp/GameProgramming/↩Heuristics.html#heuristics-for-grid-maps

5.14.3.4 Octile

```
readonly PointDiagonalCalculator CMGTGraph.Calculators.PointDiagonalCalculator.Octile [static]
```

A calculator using octile diagonal distance.

Read more on grid heuristics here: http://theory.stanford.edu/~amitp/GameProgramming/↩Heuristics.html#heuristics-for-grid-maps

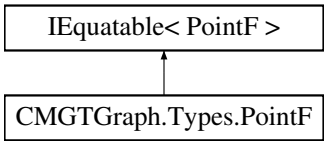
The documentation for this class was generated from the following file:

- CMGTGraph/Calculators/PointDiagonalCalculator.cs

5.15 CMGTGraph.Types.PointF Class Reference

A simple class that represents a point in 2D space with floating point values.

Inheritance diagram for CMGTGraph.Types.PointF:



Public Member Functions

- PointF (float x, float y)

  Create a floating point Point. The values are completely readonly and cannot be changed after construction.
- bool Equals (PointF other)
- override bool Equals (object obj)
- override int GetHashCode ()
- override string ToString ()

Static Public Member Functions

- static bool **operator==** (PointF left, PointF right)
- static bool **operator!=** (PointF left, PointF right)

Public Attributes

- readonly float **X**
- readonly float **Y**

5.15.1 Detailed Description

A simple class that represents a point in 2D space with floating point values.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 PointF()

```
CMGTGraph.Types.PointF.PointF (
            float x,
            float y )
```

Create a floating point Point. The values are completely readonly and cannot be changed after construction.

Parameters

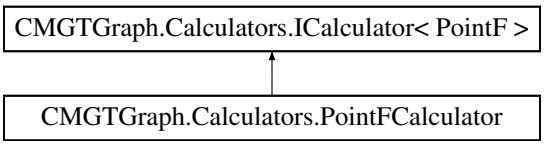
x	
y	

The documentation for this class was generated from the following file:

- CMGTGraph/Types/PointF.cs

5.16 CMGTGraph.Calculators.PointFCalculator Class Reference

Inheritance diagram for CMGTGraph.Calculators.PointFCalculator:



Public Member Functions

- float SqrDistance (PointF a, PointF b)
- float Distance (PointF a, PointF b)

Static Public Attributes

- static readonly PointFCalculator This

  The instance of this calculator.

Static Private Member Functions

- static void ThrowIfNull (PointF a, PointF b)

  Throw an ArgumentNullException if one of the provided arguments is null.

5.16.1 Detailed Description

T is in this case of type PointF. The distance calculation used here is Euclidean distance.

5.16.2 Member Function Documentation

5.16.2.1 ThrowIfNull()

```
static void CMGTGraph.Calculators.PointFCalculator.ThrowIfNull (
            PointF a,
            PointF b )  [static], [private]
```

Throw an ArgumentNullException if one of the provided arguments is null.

5.16.3 Member Data Documentation

5.16.3.1 This

```
readonly PointFCalculator CMGTGraph.Calculators.PointFCalculator.This  [static]
```

The instance of this calculator.

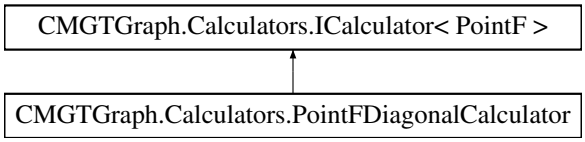
The documentation for this class was generated from the following file:

- CMGTGraph/Calculators/PointFCalculator.cs

5.17 CMGTGraph.Calculators.PointFDiagonalCalculator Class Reference

A special calculator using Diagonal distance on integer points (Point). Use this if you can move in 8 directions.

Inheritance diagram for CMGTGraph.Calculators.PointFDiagonalCalculator:



Public Member Functions

- float SqrDistance (PointF a, PointF b)
- float Distance (PointF a, PointF b)

Static Public Attributes

- static readonly PointFDiagonalCalculator Octile
- static readonly PointFDiagonalCalculator Chebyshev

Private Member Functions

- PointFDiagonalCalculator (float straight=1f, float diagonal=1.41421356237f)

Private Attributes

- readonly float _straight

  The weight for straight edges.
- readonly float _diagonal

  The weight for diagonal edges.

5.17.1 Detailed Description

A special calculator using Diagonal distance on integer points (Point). Use this if you can move in 8 directions.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 PointFDiagonalCalculator()

```
CMGTGraph.Calculators.PointFDiagonalCalculator.PointFDiagonalCalculator (
            float straight = 1f,
            float diagonal = 1.41421356237f )  [private]
```

Create a new calculator with new weights for straight and diagonal edges. The default values are 1 for straight edges and SQRT(2) for diagonal ones.

This implicitly hides the constructor with default values, to prevent creating a new instance on the outside.

5.17.3 Member Data Documentation

5.17.3.1 _diagonal

```
readonly float CMGTGraph.Calculators.PointFDiagonalCalculator._diagonal  [private]
```

The weight for diagonal edges.

5.17.3.2 _straight

```
readonly float CMGTGraph.Calculators.PointFDiagonalCalculator._straight  [private]
```

The weight for straight edges.

5.17.3.3 Chebyshev

```
readonly PointFDiagonalCalculator CMGTGraph.Calculators.PointFDiagonalCalculator.Chebyshev
[static]
```

A calculator using chebyshev diagonal distance, which basically means diagonal and straight edges have the same weight, thus diagonal edges are preferred using this calculator as a heuristic, compared to using octile distance (with Octile).

Read more on grid heuristics here: http://theory.stanford.edu/~amitp/GameProgramming/↩Heuristics.html#heuristics-for-grid-maps

5.17.3.4 Octile

```
readonly PointFDiagonalCalculator CMGTGraph.Calculators.PointFDiagonalCalculator.Octile  [static]
```

A calculator using octile diagonal distance.

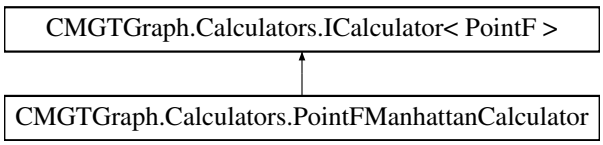
Read more on grid heuristics here: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#heuristics-for-grid-maps

The documentation for this class was generated from the following file:

- CMGTGraph/Calculators/PointFDiagonalCalculator.cs

5.18 CMGTGraph.Calculators.PointFManhattanCalculator Class Reference

Inheritance diagram for CMGTGraph.Calculators.PointFManhattanCalculator:



Public Member Functions

- float SqrDistance (PointF a, PointF b)
- float Distance (PointF a, PointF b)

Static Public Attributes

- static readonly PointFManhattanCalculator This

  The instance of this calculator.

Private Member Functions

- PointFManhattanCalculator ()

  Don't construct your own, use This!

5.18.1 Detailed Description

The type is in this case PointF
The distance is calculated using manhattan distance, which is basically the distance on both axis added together. Use this, if you can move in 4 directions (on a grid).

5.18.2 Constructor & Destructor Documentation

5.18.2.1 PointFManhattanCalculator()

```
CMGTGraph.Calculators.PointFManhattanCalculator.PointFManhattanCalculator ( )  [private]
```

Don't construct your own, use This!

5.18.3 Member Data Documentation

5.18.3.1 This

```
readonly PointFManhattanCalculator CMGTGraph.Calculators.PointFManhattanCalculator.This  [static]
```

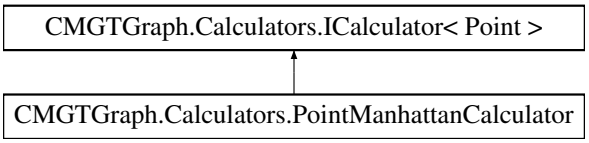
The instance of this calculator.

The documentation for this class was generated from the following file:

- CMGTGraph/Calculators/PointFManhattanCalculator.cs

5.19 CMGTGraph.Calculators.PointManhattanCalculator Class Reference

Inheritance diagram for CMGTGraph.Calculators.PointManhattanCalculator:



Public Member Functions

- float SqrDistance (Point a, Point b)
- float Distance (Point a, Point b)

Static Public Attributes

- static readonly PointManhattanCalculator This

  The instance of this calculator.

Private Member Functions

- PointManhattanCalculator ()

  Don't construct your own, use This!

5.19.1 Detailed Description

The type is in this case Point
The distance is calculated using manhattan distance, which is basically the distance on both axis added together.
Use this, if you can move in 4 directions (on a grid).

5.19.2 Constructor & Destructor Documentation

5.19.2.1 PointManhattanCalculator()

```
CMGTGraph.Calculators.PointManhattanCalculator.PointManhattanCalculator ( )  [private]
```

Don't construct your own, use This!

5.19.3 Member Data Documentation

5.19.3.1 This

```
readonly PointManhattanCalculator CMGTGraph.Calculators.PointManhattanCalculator.This  [static]
```

The instance of this calculator.

The documentation for this class was generated from the following file:

- CMGTGraph/Calculators/PointManhattanCalculator.cs