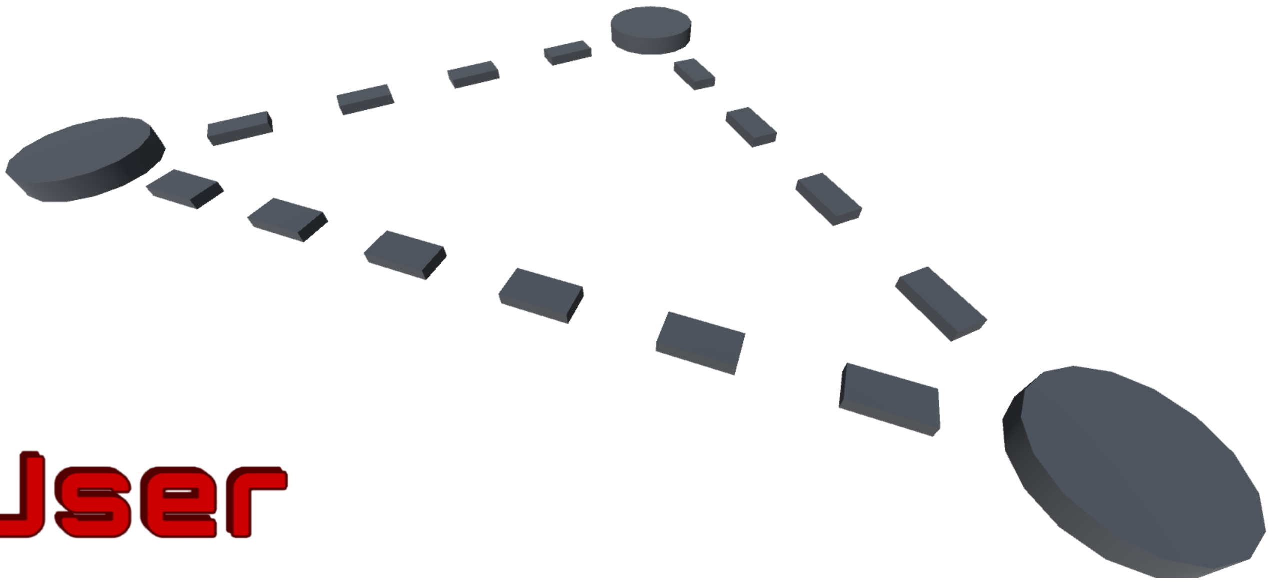
Node Map



User Manual

A Product by Justin Schneider

Version 1.5

Contents

About

Thank you for purchasing Node Map! I created this asset because I wanted to add node-based maps to a game I was making. It quickly became apparent that others would benefit from having these tools, and so I set out to make the system simple, powerful, and versatile.

I believe this tool set will get better the more people use it and provide feedback, so I ask that you not hesitate to reach out to me with feature requests, bug reports, or any other enhancements.

Installation

Installation of Node Map requires nothing more than downloading it from the Asset Store and importing it into your project. You may move the Node Map folder into any subfolder you wish.

The asset includes a demo scene located in its own folder. This folder can be safely deleted before publishing your game if you so desire.

Getting Started

Once installed, the best way to create a new Node Map is by using the menu bar:

Tools → Node Map → Create New Map

This will create a new GameObject in your scene with child objects named 'Nodes' and 'Agents' for use as containers for Nodes and Agents, respectively. If you do not use the menu bar option, you should add these objects yourself to avoid receiving errors when adding Nodes. Trust me… it's just easier using the menu option!

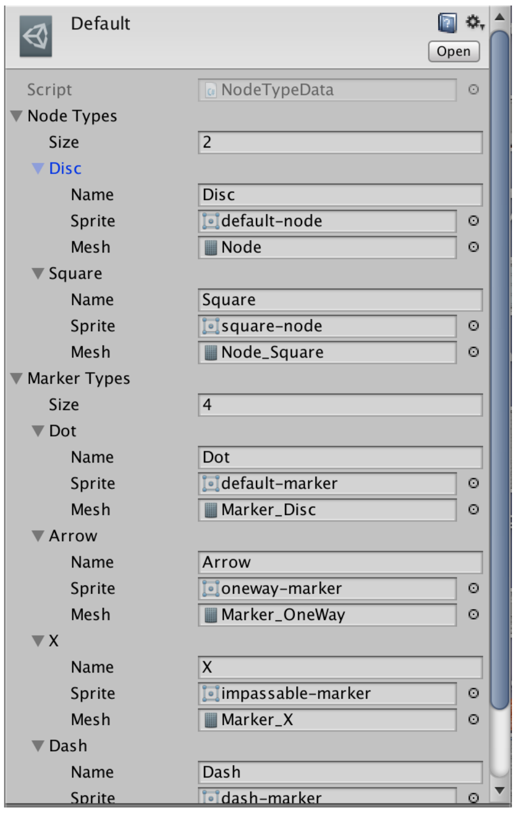
The only other thing you have to do before getting to work is to assign an instance of the NodeTypeData ScriptableObject class to the Map's *Node Data* field. This class is covered in more detail below. I've included a few pre-built options for you in the folder NodeMap/NodeTypeData. If you intend to modify these, consider copying them to a folder outside the asset's folder to avoid them being overwritten upon updates. You may also create your own NodeTypeData instances by right-clicking inside your Project view and choosing:

Create → Node Map → Node Type Data

Important

You must assign a *NodeTypeData* instance to the Map's *Node Data* field before adding Nodes. You will not be able to add Nodes or Agents to the Map without it.

Node Type Data



```
Default                                  Open
Script            NodeTypeData
▼ Node Types
    Size          2
  ▼ Disc
      Name        Disc
      Sprite      default-node
      Mesh        Node
  ▼ Square
      Name        Square
      Sprite      square-node
      Mesh        Node_Square
▼ Marker Types
    Size          4
  ▼ Dot
      Name        Dot
      Sprite      default-marker
      Mesh        Marker_Disc
  ▼ Arrow
      Name        Arrow
      Sprite      oneway-marker
      Mesh        Marker_OneWay
  ▼ X
      Name        X
      Sprite      impassable-marker
      Mesh        Marker_X
  ▼ Dash
      Name        Dash
      Sprite      dash-marker
```

NodeTypeData is a class used to describe the visual elements of a Map: its Nodes and Markers. Although used for visual purposes only in the included demo, the data may also be used in your external scripts to determine functionality.

You must have at least one Node type defined, and at least one Marker type. You should also assign a default material to be used for rendering mesh data.

Node Types

Add a new Node type here to allow the option of assigning a new appearance to Nodes in any Map that uses this data. Assigning types will be covered in more detail in the Nodes section below.

Name - The name of the type as it should appear in the Inspector when customizing Nodes
Sprite - The sprite to render when the map is in 2D mode
Mesh - The mesh to use for rendering the node in 3D mode

Marker Types

These options are identical to those of the Node Types, except used to render the Path markers (dots) between Nodes.

Materials

Additionally, there are two Material slots in a NodeTypeData instance. Mesh Material is applied to Node and Marker meshes, while Line Material is used by the Line Renderer for Paths.

Components

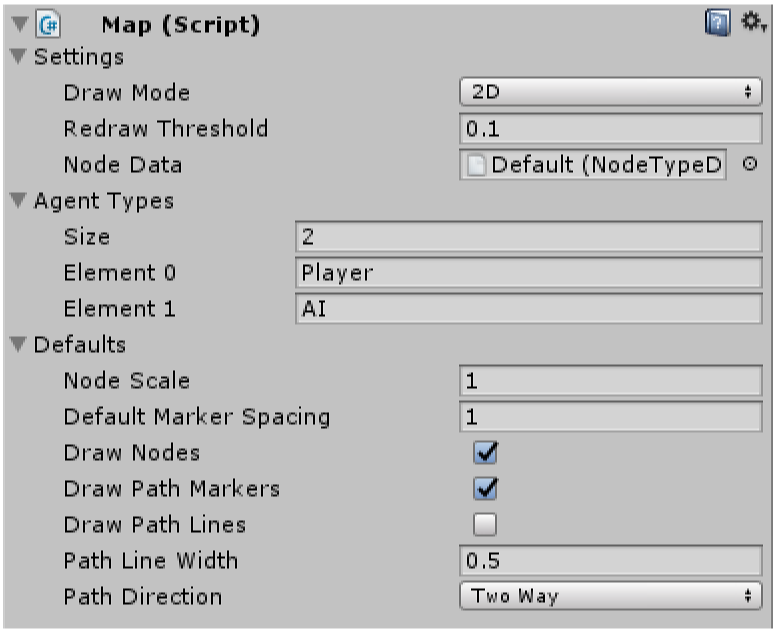
The classes provided by this asset are not intended to be added using the 'Add Component' functionality of Unity. Node Map components are hierarchical in nature, and require references to their parent Map to function correctly. This assignment, additional configuration, and proper parenting are handled for you when using the Node Map built-in controls, so that will be the advised method of working and what we will cover here.

Map

Once you've added a Map to your scene (in case you've forgotten, you do so by choosing GameObject → Node Map → Create New Map from the menu bar), adjust its settings by selecting it in the Hierarchy or Scene views.

You may have as many Maps as you like in a single Scene, but recognize that Agents are not able to travel between Maps at this time and must be copied and reassigned manually.

Settings



Map (Script)

Settings

Draw Mode	2D
Redraw Threshold	0.1
Node Data	Default (NodeTypeD

Agent Types

Size	2
Element 0	Player
Element 1	AI

Defaults

Node Scale	1
Default Marker Spacing	1
Draw Nodes	✓
Draw Path Markers	✓
Draw Path Lines	☐
Path Line Width	0.5
Path Direction	Two Way

Draw Mode - The Map may be rendered in 2D or 3D. Changing this option in the Editor will force an update of all the Map's Nodes and Markers. If changing the Draw Mode at runtime, you'll want to call the Map's `RedrawMap` method as well.

Redraw Threshold - The amount of distance a Node must be moved to update its Path/Markers in the Scene view. Increase this number to decrease the performance cost of creating and editing large Maps, and decrease it if your Map's scale causes issues with Paths not updating frequently enough.

Node Data - This is where you'll drop an assignment to those NodeTypeData instances we were talking about earlier!

Agent Types

Here you can edit the list of tags that will be used by the Map for assigning Agent types. These types will be used to set overrides for Path movement rules. Each Agent may have only one type tag, so consider your movement needs when assigning Agent types.

Another potential use of Agent types are when two Agents collide on the map. You may wish to have friendly units pass by unaffected, but enemy units initiating combat, for example. Type tags are an effective tool for this purpose.

Defaults

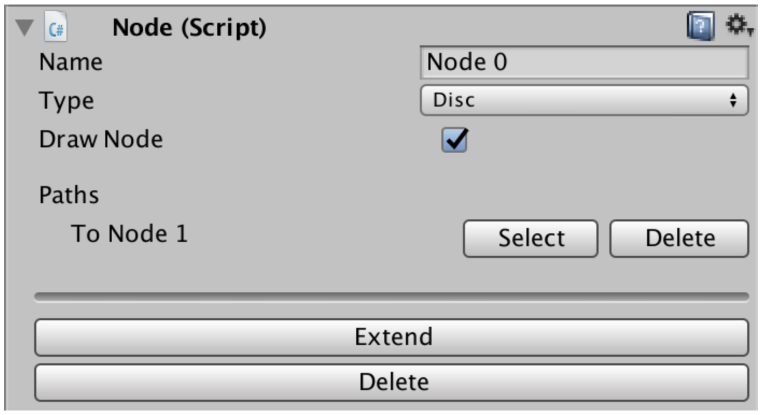
The default values listed below apply only to new Nodes. Changing these fields will not impact existing Nodes ***unless*** you change the Draw Mode or call the `RedrawMap` method (as doing either will clear the previous objects out and draw them fresh).

Node Scale - Default scale for newly created Nodes. Note that changing this value does not affect previously created Nodes.
Marker Spacing - Default spacing (in Unity units) between markers along a Path. Since Node Map will try to place markers evenly along the Path, the actual distance between markers may be larger (not to exceed twice this value).
Draw Nodes - Unchecking this box will cause new Nodes to not be rendered, useful if your Nodes' visual representation will be handled outside of Node Map.
Draw Path Markers - Default state for whether or not to render Markers along a Path
Draw Path Lines - Default state for rendering Paths as a line.
Path Line Width - Width of the line used by the Path Line Renderer. ***Note***: If you change this value, you'll have to run the Redraw Map function to update all existing paths. Otherwise, they'll use the previously entered value.
Path Direction - Default value to use for new Paths' type. Path types will be covered in more detail later.

Functions

New Node - Creates a new Node assigned to this Map. Please see the next section for more details on Nodes.
New Agent - Creates a new Agent assigned to this Map. Agents will be covered in depth later.
Redraw Map - If, for some reason, you messed with a Map's Paths and broke their rendering, or you changed some default display items that you want to revert, or something you changed doesn't seem to be reflected in the Map, this button will redraw all Nodes and Paths using the *current* default Map settings.
Refresh - Look, programming is hard. Sometimes things get a bit out of sorts. If you find yourself running into weird Null Reference exceptions that don't make sense, this button will cause the Map to search its hierarchy and refresh all if its Node and Path references. If this button goes away in the future, it's because I became a much better programmer.

Node


Node (Script)
Name: Node 0
Type: Disc
Draw Node: ✓
Paths
To Node 1

Nodes are the heart and soul of this asset. That's why I named it 'Node Map' instead of 'Something Else Map'. Nodes represent a position of importance. What exactly that means is up to you and your game's design.

Settings

Name - Obvious enough, but this is the specific Node's identifier. It's not required to be unique, but if trying to access a Node by name, only the first one found will be returned.

Type - This list is populated from the *Node Types* list in the Map's NodeTypeData. The Node will be rendered using the Sprite or Mesh associated with that type.

Draw Node - Disabling this option will cause the Node to have no visual representation on the Map. If you are not drawing the Node through Node Map, ensure your players will know through other means that they can interact with that point.

Paths - A list of all the Paths going to and from this Node. You can quickly select the desired Path by clicking the Edit button next to it, or remove the connection by clicking Delete. Paths are covered in depth in the next section.

Functions

Extend - This button allows you to quickly create a new Node at the current Node's position. The current Node and new Node will be connected by a new Path and the new Node set as the selected object. This allows you to more quickly create routes of connected Nodes without having to connect them all manually.

Delete - Self-explanatory. Say goodnight, Node! Deleting a Node will also delete any Paths that connect to it.

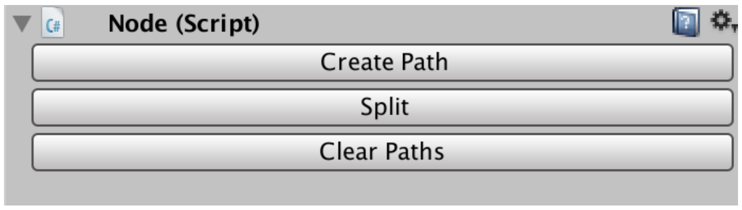
With Multiple Nodes Selected

With multiple Nodes selected, you can change the Type and Draw Node options for multiple Nodes at once, but you also have a few other options available to you.

Create Path - Create a Path connecting the selected Nodes. If more than 2 Nodes are selected, Node Map will attempt to create a route connecting all Nodes in a line.

Split - If a Path connects the selected Nodes, it will be divided into two, with a new Node joining them.

Clear Paths - Any Paths connecting the selected Nodes will be removed.



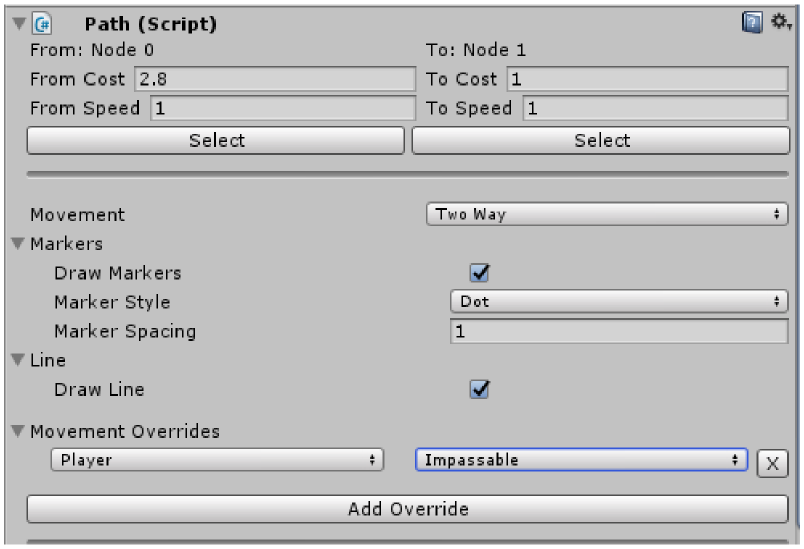
Scene View Shortcuts

In addition to the function buttons provided in the Inspector, you can also use keyboard shortcuts inside the Scene view. Note that the Scene view must be active (clicked into) in order for these to function.

CTRL+E - Extend
CTRL+J - Create Path

Path

Paths are a connection between two Nodes. As mentioned previously, Paths are created for you automatically when you Extend from an existing Node, or when you use the Create Path function with two or more Nodes selected. Since Paths are the means by which Agents can move about the Map, they are consequently the most complex part of Node Maps.


Path (Script)
From: Node 0 | To: Node 1
From Cost 2.8 | To Cost 1
From Speed 1 | To Speed 1
Movement: Two Way
Markers
Draw Markers: ✓
Marker Style: Dot
Marker Spacing: 1
Line
Draw Line: ✓
Movement Overrides
Player: Impassable

Nodes

At the top of the Path Inspector, you'll see the information for the two Nodes this Path connects. On the left is the "From" Node, and on the right, the "To" Node. Beneath each listed Node is a "Select" button to quickly select that Node and jump to its Inspector.

Cost - In pathfinding terms, how costly is this path compared to others of equal distance? This may be used if a path goes through difficult terrain that AI may want to avoid, for example. You could also access this property for deducting resources at a higher rate than normal for some Paths.
Speed - A multiplying factor for moving Agents along this path. Higher numbers increase the Agent's travel speed, while numbers closer to 0 reduce it.

Note there are separate Cost and Speed values for both Nodes. The associated value is used when traveling *starting* at that Node. In this way you can define different values for each direction of movement along a Path.

Settings

Movement - The default direction of movement allowed along a Path.
Two-Way - Agents can move in either direction
One-Way - Agents can only move from the "From" Node to the "To" Node
Reverse - Like One-Way, except can only move from "To" to "From"
Impassable - Agents may not move along this Path

Draw Markers - Whether or not to render this Path's markers
Marker Style- Uses the values in your Map's NodeData "Marker Types" list to allow you to choose a set of mesh and sprite data to render this Path
Marker Spacing - How far apart the Path's markers are
Draw Line - Whether this path should be rendered as a line

Overrides

Add an Override to a Path to allow Agents of a particular type to treat the Path differently than the default Movement type allows. You may do this to temporarily make a Path innavigable, or to disallow enemy type Agents from moving into a certain area.

You may define pre-existing Overrides by using the Add Override button. You then select an Agent type and a Movement type for this path to be calculated as for that type. To do this at runtime, see the API Reference later in this manual.

Functions

Reverse - Swap the "From" and "To" Nodes for this Path. Really only important when using the One-Way or Reverse movement rules
Split - Divide this Path into two, with a new Node between them.
Delete - Remove this Path from the plane of existence. Any connected Nodes will remain unaffected (except for having one less Path)

Marker

Markers, by default, cannot be selected or Inspected. Since they are dynamically created and destroyed as you modify your Paths, the idea is that you should not change their properties directly.

If you need to disable this feature so that you can edit Markers yourself, remove or comment out line 11 of MarkerEditor.cs.

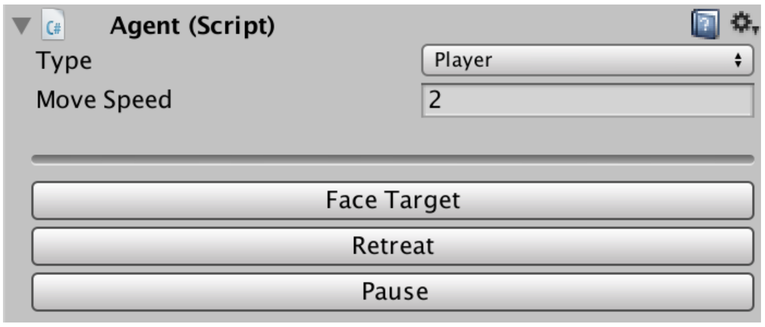
Agent

Agents are the users of your Maps. Their existence in the Node Map system is purely functional, and it is up to you to provide your own visual representation. For testing and debugging purposes, however, I've included a simple arrow object.

The recommended way of using Agents with your custom controllers is to add a new Agent using the Map's "Add Agent" button or through the API call. Make your custom controller a child of this Agent object and save a reference to the Agent component in your controller. You can then use this reference to make calls to the entire Node Map system and subscribe to movement events for that Agent.

Settings

Type - The Agent type for this Agent, defined in the parent Map.
Move Speed - This Agent's movement speed in Unity units per second. This value may be affected by a Path's Speed setting for a particular direction of movement.


Agent (Script)
Type: Player
Move Speed: 2

Movement Types

In order to suit common use cases, Node Map includes 4 movement scripts out of the box. Two of these (Click to Move and Direct Move) are intended for player-controlled Agents, and the others two (Patrol and Wander) for AI-controlled units.

To use these scripts, attach one to an existing Agent.

Click to Move

This movement type allows you to click or tap on a Node, and any active Agents with this script will attempt to move to that Node. Note that input for this script will not be processed while the Agent is moving.

Active - Whether or not this Agent should respond to clicks. Use this in conjunction with your own selection logic to enable/disable selected Agents for moving multiple entities around the Map
Max Distance - The maximum distance a raycast from the camera should look for Nodes

Direct Move

This type allows Agents to be controlled with direct keyboard or controller input using the X and Y axes. As above, input is not processed while the Agent is moving.

Active - Whether or not this Agent should respond to keyboard/controller input.
Detection Angle - How precise the angle of input needs to be. Smaller values require more precise input, while larger values allow more leeway.
Dead Zone - Input vectors below this value will be ignored. Loose joysticks may not center at 0,0.
Input Delay - How long after input has been received to process movement vector. Keyboard input can be a bit tricky for diagonal Paths, so giving a bit of extra time can reduce unintended movements.
Move Ref - Should input be relative to the global axes (Up = +Z, Right = +X), or relative to the camera (Up = Screen position up, Right = Screen position right). You *probably* always want this to be Camera.

Patrol

This movement script causes an Agent to move along a set of target Nodes. Upon reaching the last Node in its list, it will return to the first Node and start again.

Active - Dictates if this Agent should continue its patrol
Waypoints - List of Nodes to travel between
Current Point Index - The index of the Node the Agent is currently on

Wander

This script causes the Agent to randomly select a Node from the Map and attempt to move to it. Once it reaches its destination (or the destination is determined to be unreachable), it will select a new random Node and continue the process.

Active - Enables or disables the wandering behavior.

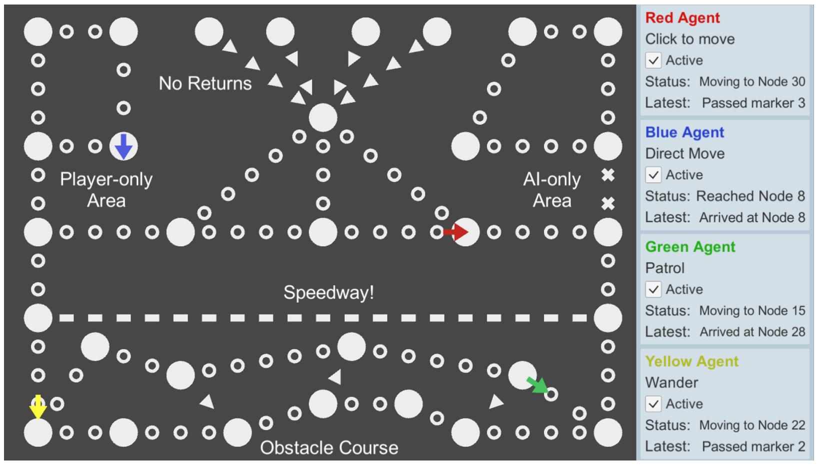
Demos

There are two included demo scenes showing some examples of the various features in Node Map.

MovementDemo

This demo shows how the four included Movement scripts work, as well as demonstrates a few different settings for Paths.

- A network of Nodes and Paths!
- Path movement types for two-way, one-way, and impassable sections
- Overrides for the Player-only and AI-only area entrances
- Increased move speed and reduced cost for the "Speedway"
- The 4 included Agent movement scripts are all demonstrated
- UI panels subscribe to Agent events for displaying status



Red Agent
Click to move
✓ Active
Status: Moving to Node 30
Latest: Passed marker 3

Blue Agent
Direct Move
✓ Active
Status: Reached Node 8
Latest: Arrived at Node 8

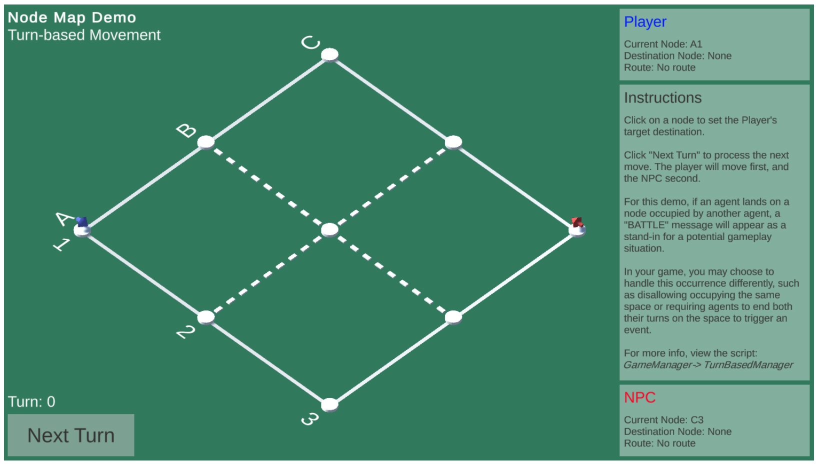
Green Agent
Patrol
✓ Active
Status: Moving to Node 15
Latest: Arrived at Node 28

Yellow Agent
Wander
✓ Active
Status: Moving to Node 22
Latest: Passed marker 2

TurnBasedDemo

This demo shows how you might use the API to incorporate your own movement types not covered by the included Movement scripts. The scene includes an object called GameManager, with a script TurnBasedManager.cs attached. All of the logic to have the Agents move in a turn-based manner is located in this script.

It should be noted that the included script is a naive implementation for demonstration purposes only and should not be used as-is. Rather, you should use this script for guidance on how you might use the same methods in your own setup.



Player
Current Node: A1
Destination Node: None
Route: No route

Instructions
Click on a node to set the Player's target destination.

Click "Next Turn" to process the next move. The player will move first, and the NPC second.

For this demo, if an agent lands on a node occupied by another agent, a "BATTLE" message will appear as a stand-in for a potential gameplay situation.

In your game, you may choose to handle this occurrence differently, such as disallowing occupying the same space or requiring agents to end both their turns on the space to trigger an event.

For more info, view the script:
GameManager-> TurnBasedManager

NPC
Current Node: C3
Destination Node: None
Route: No route

API Reference

Map

Static Methods

Signature	Returns
FindValidPath(Node a, Node b) Given two Nodes, returns the `Path` connecting them or `null` if no connections are found. Note that this method does not take pathfinding elements such as Movement Type or Overrides into account.	Path
GetPointOnPath(Path path, float normalizedDistance) Gets a position along a Path at normalizedDistance, with 0 being the Path's From Node position and 1 being the Path's To Node position. Used in rendering Path markers.	Vector3
RemovePath(Path path) Deletes a Path and clears out its references in the Nodes it previously connected.	void
RemoveNode(Node node) Deletes a Node along with any Paths this node is a connection for.	void

Public Methods

Signature	Returns
ClearPathsToNode(Node clearNode) Removes any Paths that have `clearNode` as a "To" connect point.	void
CreateAgent(Node node = null) Creates a new Agent for this map and sets its initial position to node. If no Node is passed in, the position of the first found Node is used.	Agent
CreateNode() Creates a new Node and adds it to the Map.	Node

GetNodeByName(string name) Searches the map for the first Node with name `name` and returns it, or `null` if none are found.	Node
GetPathsToNode(Node node) Returns an array of Paths that have `node` as a "To" node. For accessing a Node's paths "from" it, use the Node's `GetPaths` method.	Path[]
GetNodeTypeIndexByName(string typeName) Gets the index for the Node type with name `typeName`.	int
GetMarkerTypeIndexByName(string typeName) Gets the index for the Marker type with name `typeName`.	int
GetAgentTypeIndexByName(string typeName) Gets the index for the Agent type with name `typeName`.	int
IsNodeType(Node node, string typeName) Returns true if `node` is of type `typeName`.	bool
IsMarkerType(Marker marker, string typeName) Returns true if `marker` is of type `typeName`.	bool
IsAgentType(Agent agent, string typeName) Returns true if `agent` is of type `typeName`.	bool
RedrawMap(bool drawModeChanged) Redraws the Map's Nodes and Paths. If drawModeChanged is true, Nodes are instructed to update their Sprite or Mesh rendering components as appropriate.	void
Refresh() Forces the map to update its references to all Nodes and Paths. If you are adding or removing Nodes/Paths through your code, it's a good idea to call Refresh afterward to ensure all references are up to date.	void
PauseAll() Pauses movement for all Agents on map.	void

PlayAll() Resumes movement for all Agents on map.	void
HighlightRoute(List<Node> route) Given a list of Nodes, this method uses the Highlight material from a map’s Node Type Data to highlight all Nodes and Paths for the route.	void
ClearRouteHighlight() This method clears the highlight on any Nodes and Paths in a map, assigning the default materials from the Node Type Data.	void

Node

Public Methods

Signature	**Returns**
Initialize(Map nodeMap) Configures Node with proper settings from parent `nodeMap`. When adding your own Nodes via script, call Initialize with the Map reference to ensure it is setup correctly.	void
CreatePath(Node toNode, bool drawPath) Creates and returns a Path from this Node to `toNode`. If `drawPath` is true, the path’s markers will default to being drawn.	Path
GetPaths() Returns an array of all Paths with this Node as its “From” point.	Path[]
GetAllPaths() Returns an array of all Paths with this Node as its “From” or “To” point.	Path[]
GetAgentsAtNode() Returns a List of Agents currently occupying this Node.	List<Agent>
GetClosestNode([Agent agent], [bool disallowOccupied], [string type]) Gets the closest Node connected to this Node. If `agent` is not null, limits selection to paths navigable by that Agent’s type. If `disallowOccupied` is true (defaults to false), will not return Nodes that have an occupant. If `type` is not null, limits results to only those of that particular Node type.	Node

Redraw() Instructs the Node to be redrawn, typically because its type was changed.	void
SetDrawMode(DrawMode drawMode, NodeTypeData data) Method for redrawing Node when the map's DrawMode has changed.	void
SetVisible(bool isVisible) Toggles the visibility of this Node.	void

Path

Public Methods

Signature	**Returns**
Initialize(Map nodeMap, Node fromNode, Node toNode) Sets configuration options for a new Path. If adding Paths in your scripts, be sure to call this method to ensure all assignments are properly made.	void
IsValid() Returns false if either "To" Node or "From" Node are null. Otherwise returns true.	bool
RedrawMarkers() Destroys and redraws Path's markers. Used by the Editor when moving Path endpoints around.	void
Reverse() Swaps the "From" and "To" Nodes and changes Path ownership as appropriate, effectively changing the direction of the Path's movement.	void
SetVisible(bool status) Sets the visible state for this Path's markers.	void
GetOtherEnd(Node node) Helper method for quickly finding a Path's opposite endpoint. If "To" point passed in, "From" is returned, and vice versa.	Node

AddOverride(int agentType, MovementType movementType) Given an Agent Type index and a MovementType, creates a new Override for this Path if no Override existed for that Agent type. If that Agent already had an Override registered, its Movement Type is updated with the new value.	void
RemoveOverride(int agentType) Removes any Override for the Path with the given Agent type index.	void

Agent

Public Methods

Signature	**Returns**
Initialize(Map nodeMap, Node startingNode) Sets configuration options for a new Agent. When adding agents in your own scripts, call this method to ensure it's setup is completed correctly.	void
SetCurrentNodeByName(string nodeName) Assigns the Node with nodeName `nodeName` to the Agent's `currentNode` property. Does not actually move the Agent object.	void
FaceDir(Vector3 dir) Turns the Agent to look at `dir`.	void
GetRoutePointPositions(Node targetNode) Returns a list of points, starting with the Agent's current position, along the path to `targetNode`. Useful for rendering a path an Agent will take or is taking to reach its destination, using your own technique.	List<Vector3>
MoveToTarget(Node targetNode) Finds a suitable route to `targetNode` and begins moving towards it. For more details on working with moving Agents, see the Events section below.	void
MoveToTarget(string targetName) Searches for a Node named `targetName` and begins moving towards it. For more details on working with moving Agents, see the Events section below.	void

DoRetreat() If current Path would allow Agent to move in opposite direction of current travel, cancels movements towards current target, starts moving back towards previous Node. If current Path is not able to be traveled in opposite direction, retreat does not occur.	void
JumpToNode(Node targetNode) Instantly moves Agent to `targetNode`, bypassing pathfinding and spacetime limitations.	void
Pause() Pauses movement for this Agent.	void
Play() Resumes movement for this Agent.	void

Events

OnMoveStart(Node targetNode) Triggered when Agent has begun moving towards a destination Node.
OnNodeArrive(Node reachedNode, bool isTargetNode) Triggered when Agent arrives at any Node. If `reachedNode` is the Agent's target Node, `isTargetNode` will be true.
OnMoveEnd(Node targetNode) Triggered when Agent has finished moving, whether by reaching its target Node or because the Node is determined to be unreachable.
OnRetreat(Node previousNode) Triggered when the Agent has begun a valid retreat to its previous Node.
OnCannotReach(Node targetNode) Triggered when pathfinding fails to find a valid route to the Agent's target Node.
OnAgentCollide(Agent otherAgent) Triggered when this Agent comes into contact with another Agent.
OnMarkerTick(int markerIndex) Triggered when this Agent passes over a Path marker.

OnPause() Triggered when this Agent has had movement paused.
OnResume() Triggered when this Agent resumes moving from being paused.

Pathfinding

Static Methods

Signature	**Returns**
GetNeighbors(Node node, Agent agent = null) Returns a list of all Nodes that have movable connecting Paths with `node`. Movable Paths are ones that have defined Movement types allowing traversal in the direction away from `node`. If `agent` is not null, the Path's Overrides are taken into account as well.	List<Node>
AgentCanMoveAcross(Path path, Node node, Agent agent) Given a Path, the starting Node, and an Agent, returns whether or not this move is valid. If `agent` is null, Overrides are ignored.	bool
FindRoute(Node startNode, Node goalNode, Agent agent = null) Returns a list of Nodes from `startNode` to `goalNode`. If `agent` is not null, Overrides are taken into consideration.	List<Node>

Change Log

1.5

- New maps are now created under the Tools menu (Tools → Node Map → Create New Map) instead of under the GameObject menu.
- Added methods to highlight route
- Added highlight material to Node Data
- Added turn-based demo to show example of using API to create custom movement type
- Added helper method to Agent to set current node by name
- Fixed order of execution of Agent events OnMoveEnd and OnNodeArrive

1.4.1

- Uploaded version for Unity 2020.
- Updated links to documentation and demo

~~1.3~~ 1.4

- Version number skipped to match Asset Store version
- Clarified proper use of Node Scale (now used as a default rather than updating existing Nodes)
- Added logic to remove references to manually deleted Nodes and Paths
- Added Scene handles for Nodes (rather than using default behavior of centering handles on object and all children positions)

1.2

- Added new helper methods:
  - Map
    - GetNodeTypeIndexByName
    - GetMarkerTypeIndexByName
    - GetAgentTypeIndexByName
    - IsNodeType
    - IsMarkerType
    - IsAgentType
  - Node
    - GetAgentsAtNode
    - GetClosestNode
  - Agent
    - GetRoutePointPositions
- Modified properties of Agent class:
  - currentNode now points to the Node an Agent is **currently** occupying
  - lastNode points to the most recent Node an Agent successfully occupied

1.1

- Added option to render Paths as Lines
- New objects are instantiated as Static to improve runtime performance
- Changed Create Path keyboard shortcut from **CTRL+P** to **CTRL+J** to avoid conflicting with Windows “Play” editor shortcut.

1.0

- Initial release.

Contact

Unity Forum

I'm a big supporter of the community, and I strongly believe in public communication for the benefit of others. To that end, if you have problems with this asset, questions about functionality, new feature requests, comments or criticisms, please consider posting in the asset's official forum thread, found at the link below.

Additionally, I'd be delighted to see screenshots, GIFs, or video of content you've created using Node Map! If your game is using my asset, I'll do whatever I can to help promote it. It's a win-win!

Unity Forums Node Map Thread

http://bit.ly/jsnodemap

Email

If you're not the community type and prefer one-on-one communication, you can still contact me by email.

Node Map Support Email

developer@justinschneider.com