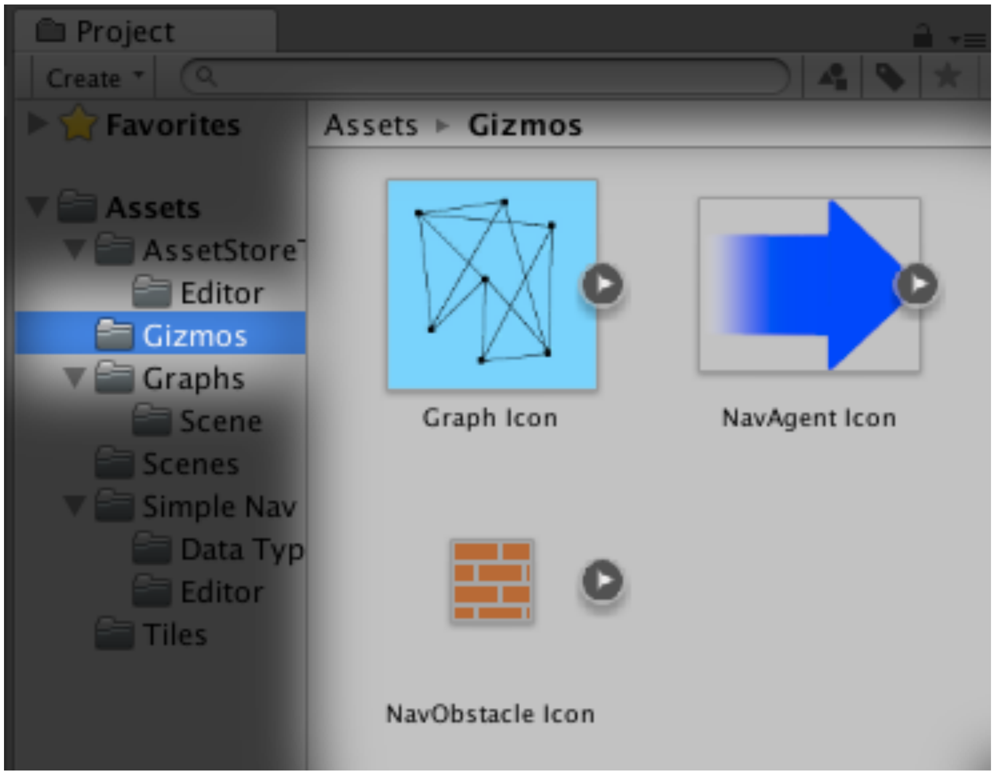
Introduction

Thank you for choosing Simple Nav! Simple Nav is a lightweight solution to AI navigation for top-down and orthographic games in Unity 2D. Simple Nav provides a system that is comparable to that provided by Unity for 3D games. Making an agent navigate intelligently around your scene is a simple 4-step process:

1. Attach Nav Obstacle components to all obstacles in the scene
2. Use the Simple Graph window to produce a graph asset file
3. Attach a Nav Agent component to the agent that will use the graph
4. Drag and drop the graph asset into the graph field of the Nav Agent component

Gizmos

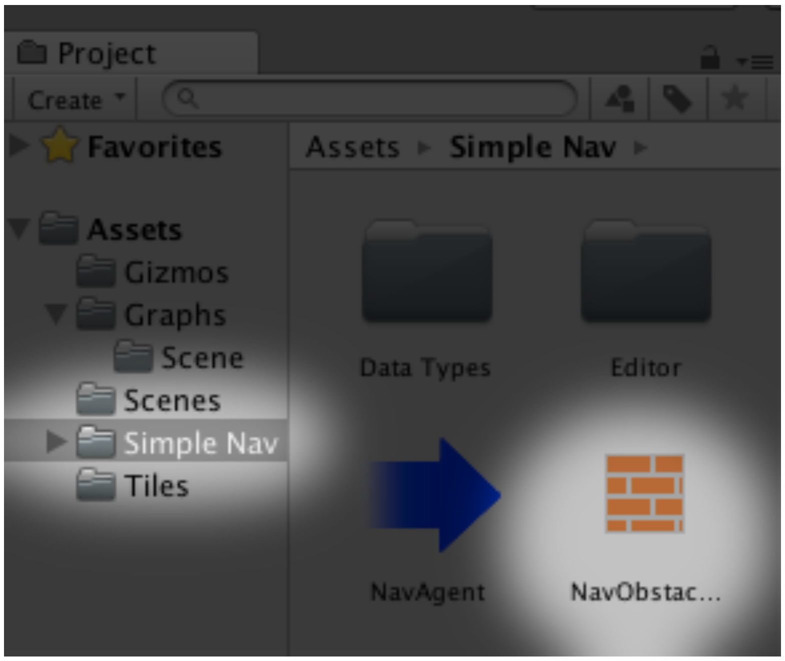
Before anything else, create a folder in the root of your assets folder named Gizmos. Locate the files in the Simple Nav package named *Graph Icon.png*, *NavAgent Icon.png*, and *NavObstacle Icon.png* and drag them into the Gizmos folder.



Icon files inside Gizmos folder

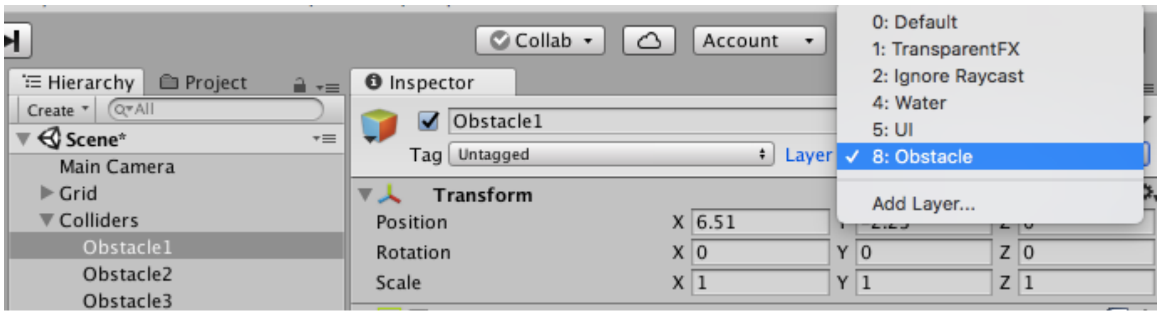
Nav Obstacle Component

Nav Obstacle components are the most import tool in building a graph that can be navigated by Nav Agents. Once a Nav Obstacle component script has been added to a game object, it is then designated as a Nav Obstacle. A Nav Obstacle object must contain two things; a Nav Obstacle component and a box collider 2D component. The box collider 2D is referenced in graph vertex placement so the collider must be fitted over the section it represents. The Nav Obstacle component script can be found in the root directory of the Simple Nav package sporting a brick wall icon.



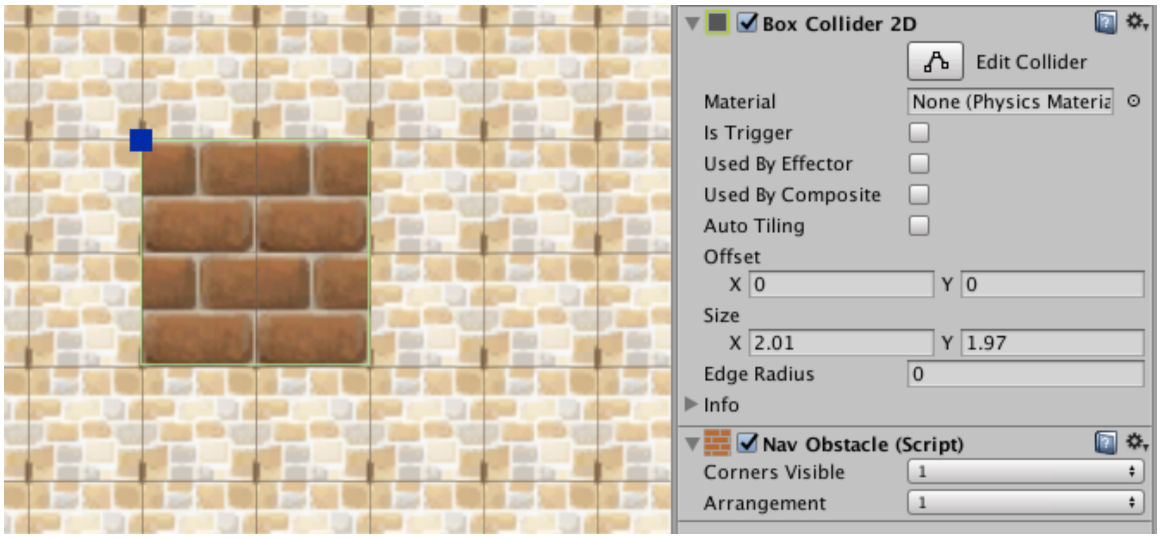
Nav Obstacle location

When a game object is designated as a Nav Obstacle, it is automatically placed on layer 8 of the scene. It is important that all boundaries in the scene be placed on layer 8 as this is critical to accurate graph construction.



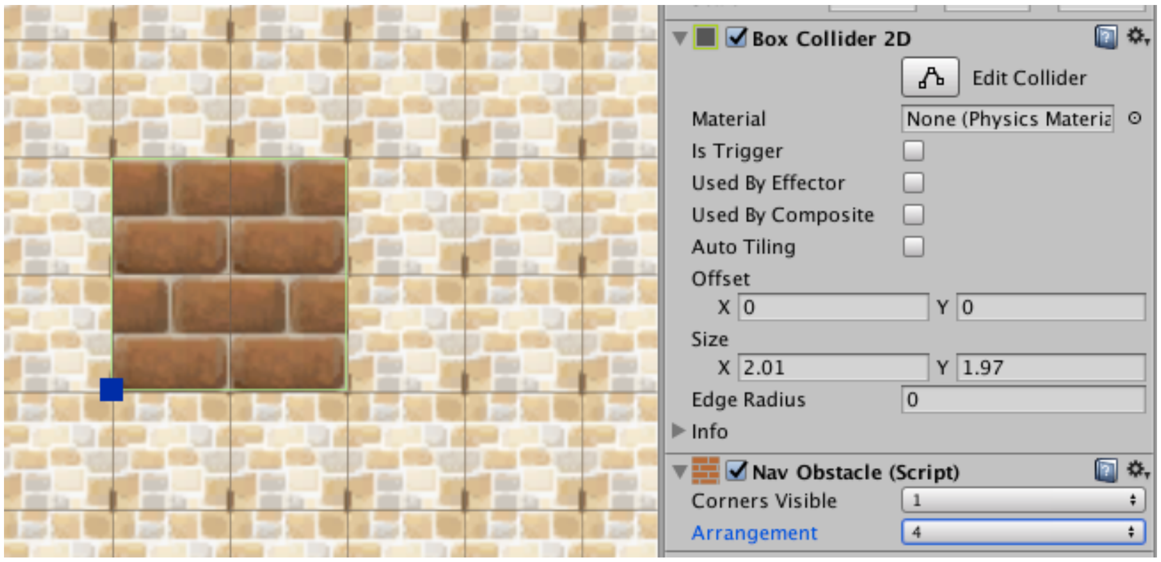
Game objects designated as Nav Obstacles will automatically place themselves on layer 8

The Nav Obstacle component provides two main settings for the user; *corners visible* and *arrangement*. The corners visible parameter specifies the number of corners on the game object's box collider 2D component that will be used in the construction of the graph of your scene. The arrangement setting allows the user to decide which corners will be used. When calculating the position of vertices around the obstacle, the arrangement setting tells the Nav Obstacle component at which corner it should begin adding vertices starting at the top left and rotating clockwise to the bottom left. The corner selection options are demonstrated with blue markers in the scene view and are updated according to the user's currently selected options.



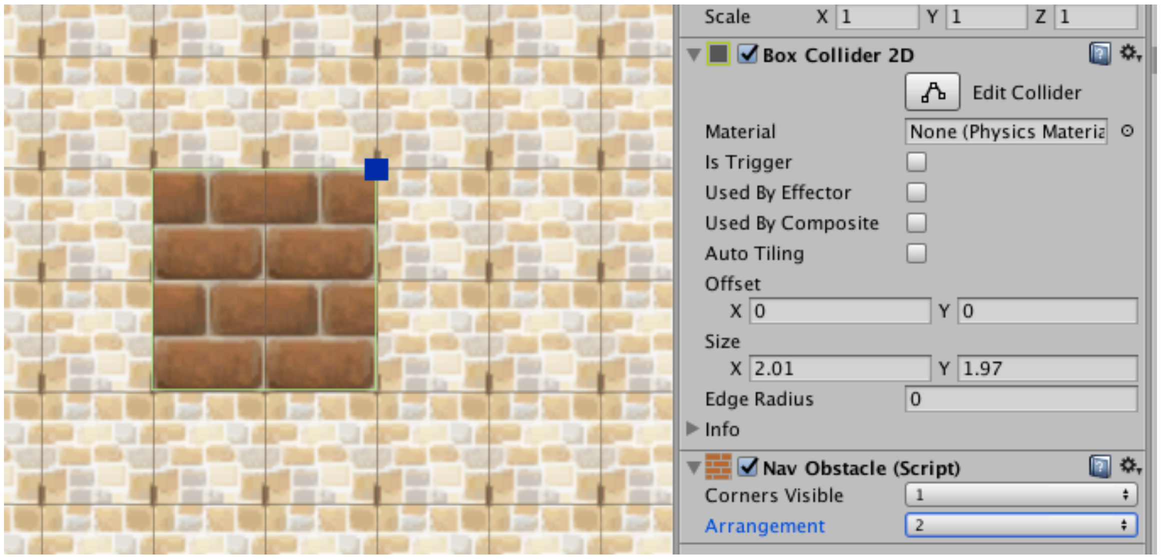
Material None (Physics Materia
Is Trigger
Used By Effector
Used By Composite
Auto Tiling
Offset X 0 Y 0
Size X 2.01 Y 1.97
Edge Radius 0
Info
Corners Visible 1
Arrangement 1

Arrangement 1



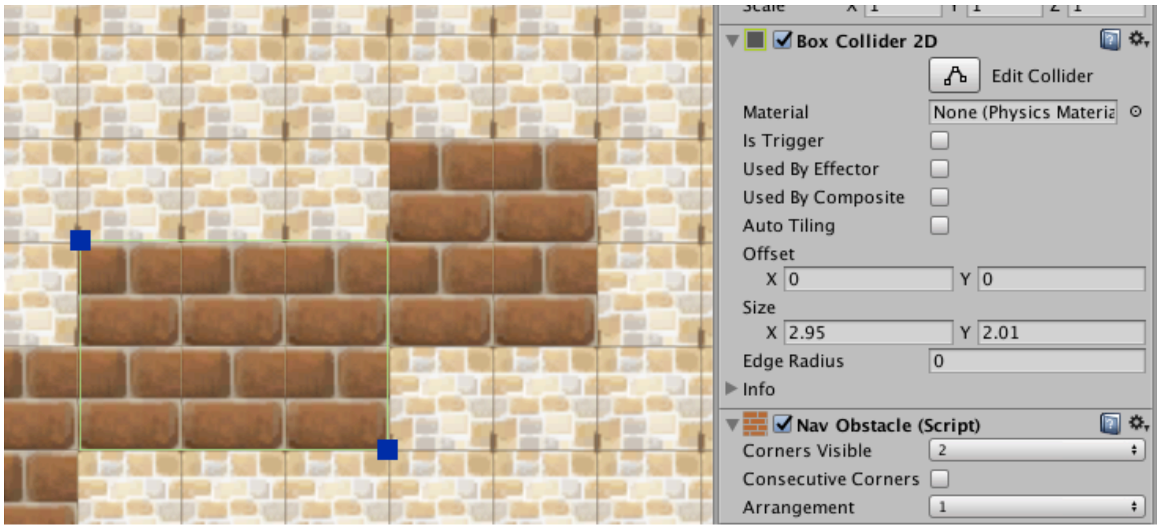
Material None (Physics Materia
Is Trigger
Used By Effector
Used By Composite
Auto Tiling
Offset X 0 Y 0
Size X 2.01 Y 1.97
Edge Radius 0
Info
Corners Visible 1
Arrangement 4

Arrangement 4


Scale X 1 Y 1 Z 1
Box Collider 2D ✓
Edit Collider
Material: None (Physics Materia
Is Trigger: ☐
Used By Effector: ☐
Used By Composite: ☐
Auto Tiling: ☐
Offset: X 0 Y 0
Size: X 2.01 Y 1.97
Edge Radius: 0
Info
Nav Obstacle (Script) ✓
Corners Visible: 1
Arrangement: 2

Arrangement 2

The options available for vertex placement vary depending on the number of corners visible. If all four corners are visible, for example, there is only one possible arrangement of vertices so the arrangement selector is not available. There is a special setting available when there are two corners visible; *Consecutive Corners*. This provides an option to place vertices on diagonal rather than adjacent corners of the obstacle.

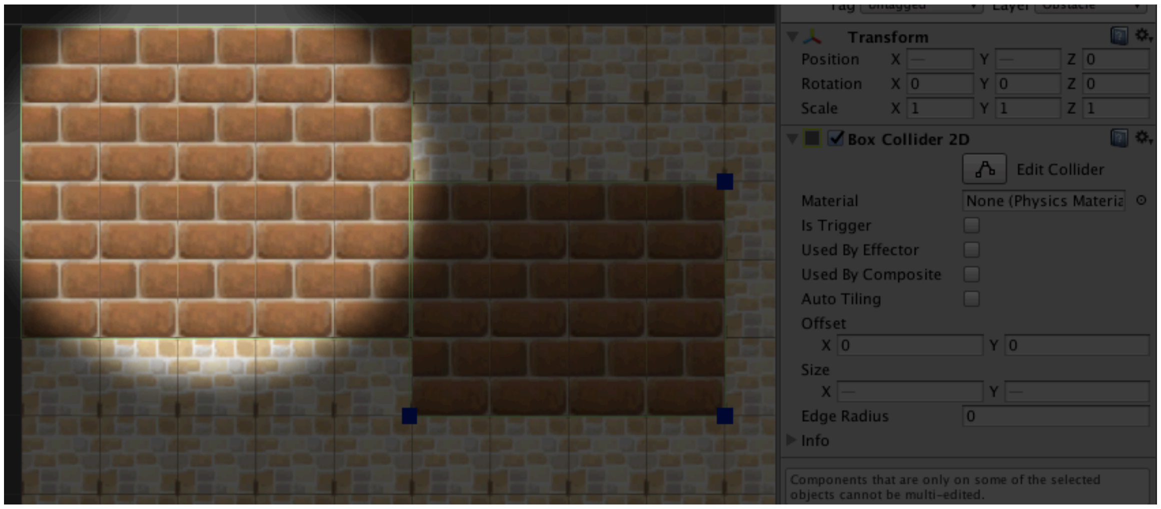

Box Collider 2D ✓
Edit Collider
Material: None (Physics Materia
Is Trigger: ☐
Used By Effector: ☐
Used By Composite: ☐
Auto Tiling: ☐
Offset: X 0 Y 0
Size: X 2.95 Y 2.01
Edge Radius: 0
Info
Nav Obstacle (Script) ✓
Corners Visible: 2
Consecutive Corners: ☐
Arrangement: 1

An example use of non-consecutive corners

In the case of non-consecutive corners, there are only two possible arrangement options.

Some objects in your scene may serve as obstacle boundaries but may not require a Nav Obstacle component since they do not contain any corners to be navigated around. Such objects include level boundaries and boundaries in corners.

Again it is required that all boundaries in the scene be placed on layer 8; even those not designated as Nav Obstacles.

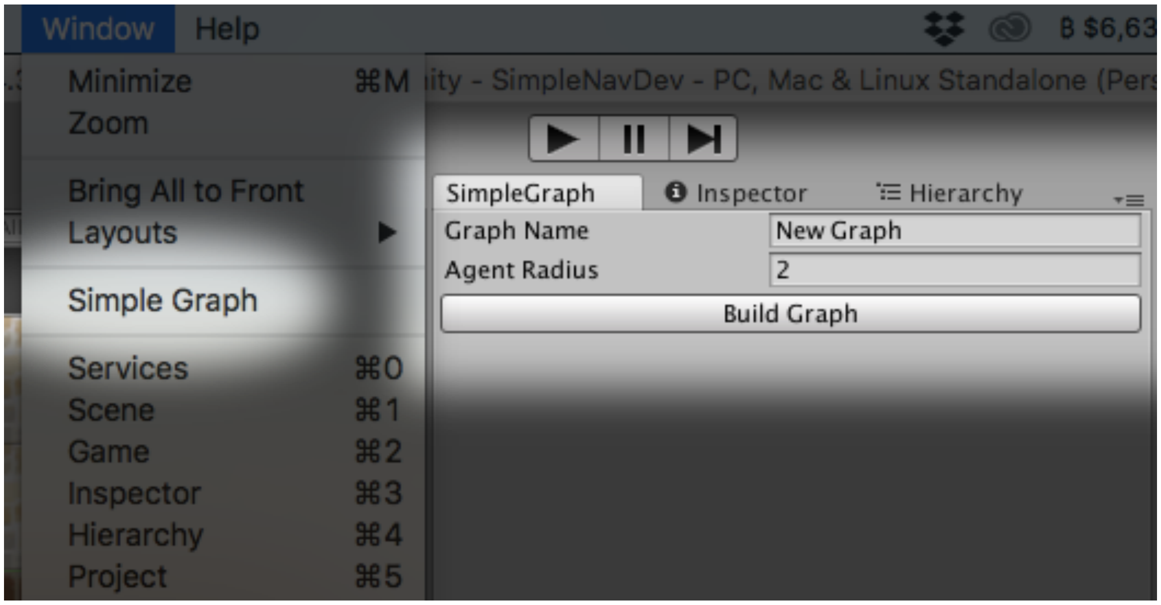


```
Transform
Position      X —     Y —     Z 0
Rotation      X 0     Y 0     Z 0
Scale         X 1     Y 1     Z 1

[✓] Box Collider 2D
                      Edit Collider
Material              None (Physics Materia
Is Trigger            [ ]
Used By Effector      [ ]
Used By Composite     [ ]
Auto Tiling           [ ]
Offset
   X 0                Y 0
Size
   X —                Y —
Edge Radius           0
Info

Components that are only on some of the selected
objects cannot be multi-edited.
```

An example of a boundary object that does not require a Nav Obstacle component

Simple Graph Window

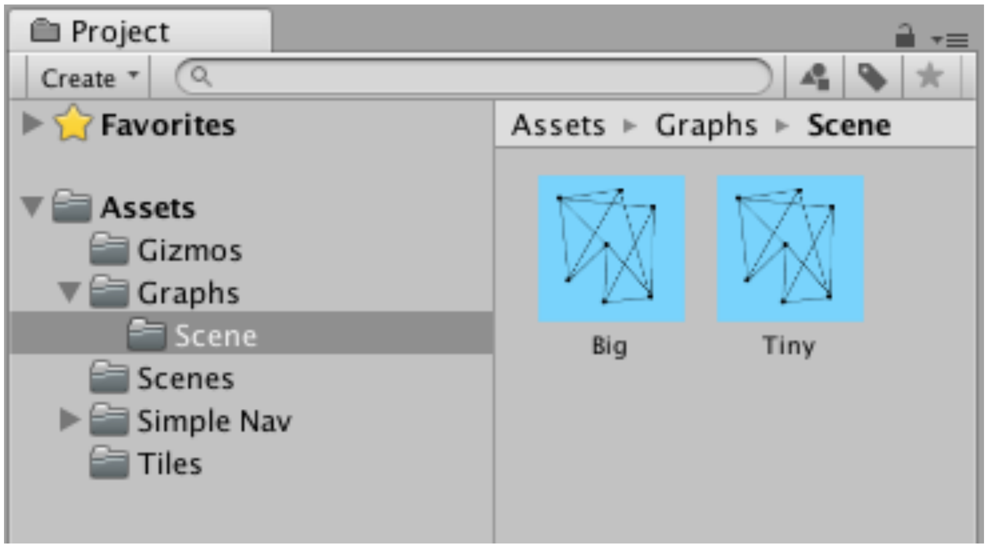
After designating objects in your scene for your agents to navigate around using Nav Obstacles, it's time to make some graphs! In the Unity Editor, go to Windows > Simple Graph to open the Simple Graph window.



Graph Name: New Graph
Agent Radius: 2

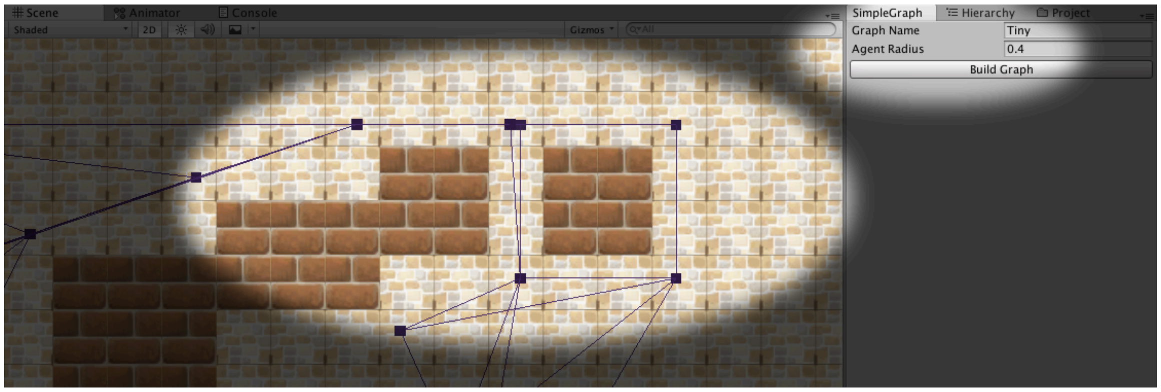
Simple Graph in the Unity window menu and open Simple Graph tab

Simple Graph is the tool used to create graph assets to be used by nav agents. Before creating a graph, a graph name and an agent radius must be provided. The graph name is the name of the graph asset file to be created. Graph names must contain non-empty characters. The agent radius is simply the radius of the circle collider attached to the game object that will be using the graph. Upon pressing *Build Graph* a graph asset folder with the given graph name will be created in the Assets/Graphs/[Current Active Scene Name] folder. If a Graphs folder does not currently exist in your Assets folder, one will be created along with a sub-folder with the name of the currently active scene.



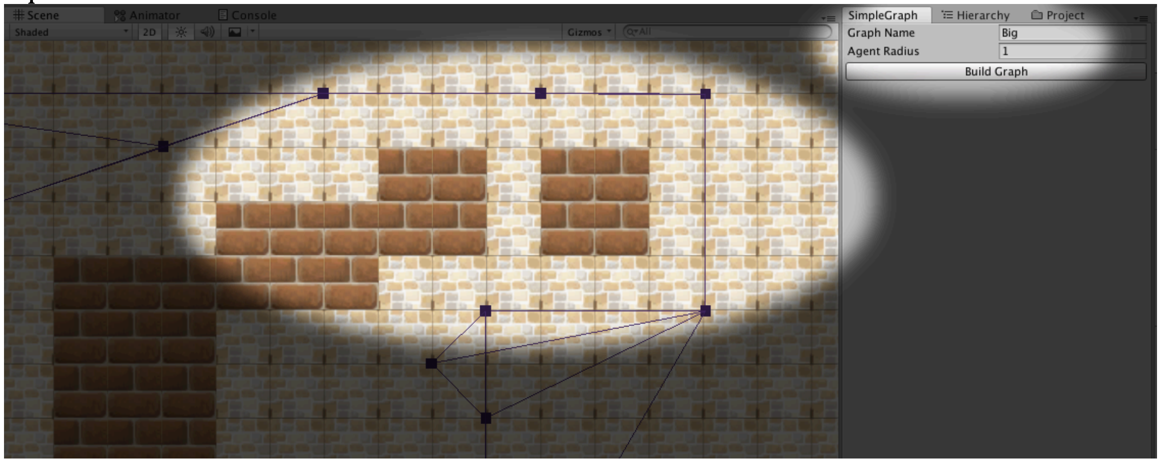
The Graphs folder. In this project the active scene is named “Scene” and two different graph assets currently exist for the scene; *Big* and *Tiny*.

There are two reasons to create more than one graph for a scene. Creating a graph specific to an agent or class of agents allows these agents to get closer to corners and navigate the scene more naturally. In addition, parts of the scene may not be navigable to agents whose radii are too large. For example, observe the following two graphs:



Graph Name: Tiny
Agent Radius: 0.4

In this graph the agent radius is small enough that it can navigate between the two square obstacles.

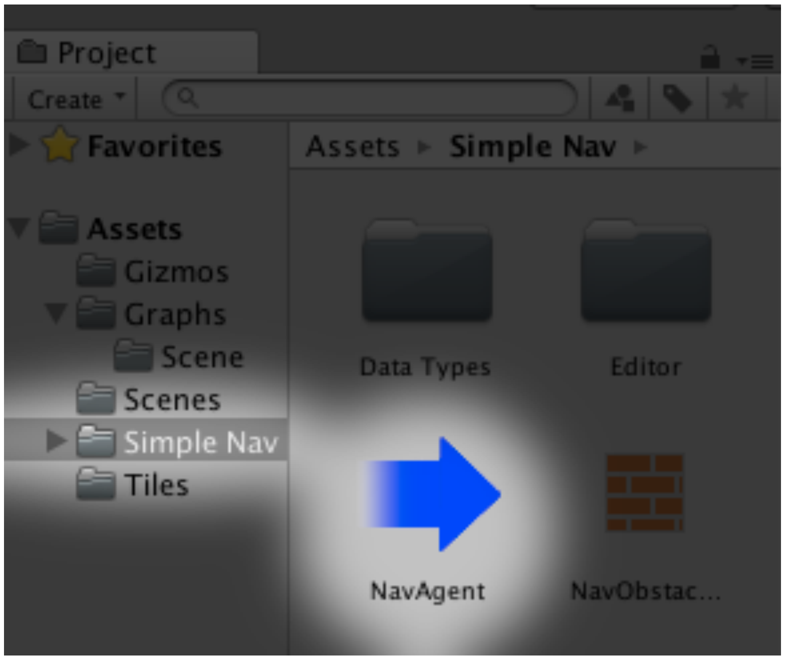


Graph Name: Big
Agent Radius: 1

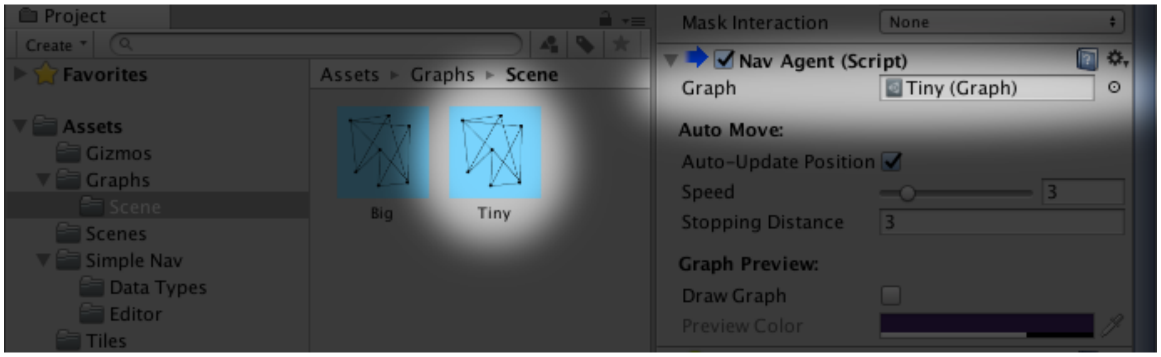
The same gap is not navigable by this agent since its radius is too large. Thus, this particular agent will be forced to go around. Also observe that the vertices in the graph with the smaller radius are closer to the obstacles than those in the *Big* graph. Since the Nav Agent(s) using the *Tiny* graph are smaller, their centers can get closer to obstacles.

Nav Agent Component

Now that we're ready to go with some graph(s), it's time to get to the fun part; making agents that can navigate your scene! To make a game object into a Nav Agent, simply add the Nav Agent component script. The Nav Agent component located in the Simple Nav root directory - like the Nav Obstacle component – and sports a stylish arrow icon.



The first thing to do after adding the Nav Agent component to an object is to give it a graph to use. Navigate to the folder where the graphs for the scene you are working in are stored and drag and drop the appropriate Graph asset into the top-most field in the Nav Agent GUI labeled "Graph".



Mask Interaction: None
Graph: Tiny (Graph)
Auto-Update Position: ✓
Speed: 3
Stopping Distance: 3
Draw Graph: ☐
Preview Color

A warning will appear in the GUI prompting the user to add a graph if one is not currently loaded. After loading a graph, a preview of the loaded graph will appear in the scene view if the *Draw Graph* option is selected.

Unlike the Nav Obstacle component, the Nav Agent component contains public methods and attributes intended to be accessed from other scripts. There are two main methods for use in the Nav Agent class; *SetDestination()* and *CheckPath().*

SetDestination takes a *Vector3* object representing the position in the scene where the agent should go as its only argument. Upon calling this method, the Nav

Agent component will calculate the shortest path to the provided destination using the A* path finding algorithm on the provided graph asset. This path will then be stored as a stack of *Vector3* objects in the public variable *path*. If a path cannot be created for whatever reason, the pathExists variable will be set to false and path will contain an empty stack.

CheckPath also takes a destination as a Vector3. It is used to check if it possible to find a path to the destination before setting a new destination.

If the user has the *Auto-Update Position* option checked, the agent will automatically move to its destination if a path can be found. The *Speed* variable determines the agent's rate of travel. The *Stopping Distance* variable with no further action required. The second way allows the developer to move the agent manually however they would like using the path that has been generated.

Most of the time, a user will also want to update their agent's animator component to animate their movement. This is done using the Nav Agent component's public *Vector3* variable; *currentDirection*. *currentDirection* is the direction the agent is currently moving and can be used to update the animator.