Customizable Game Grid Documentation

by TechnOllieG

0.1-beta

Table of contents

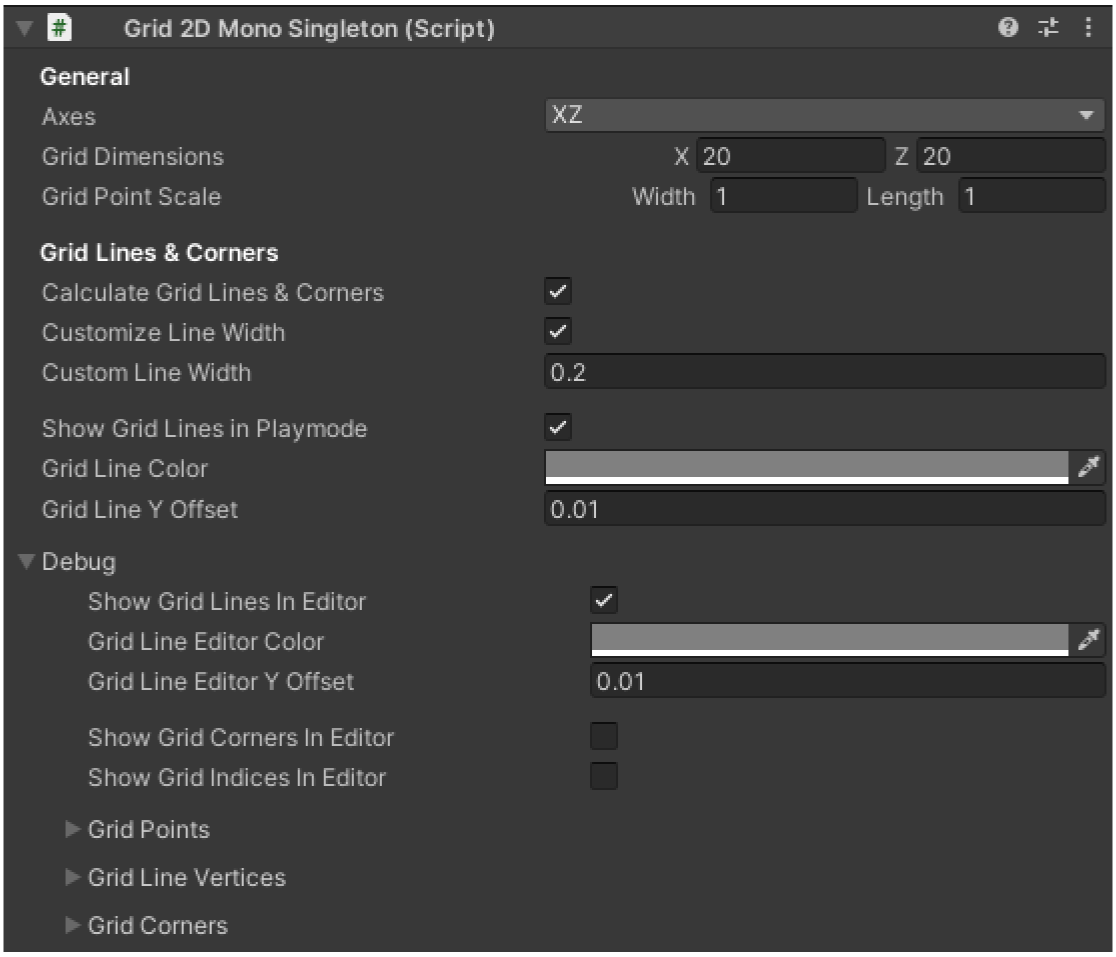
Overview

Showcase video: https://youtu.be/muk9A261jYs

The customizable game grid is a tool for Unity that allows the user to create grids to be used in games. Grids exist at its core as standard C# classes with an array of grid points, the amount of which is determined by the width/length of the grid. Each point being uniformly spaced out based on the grid's origin and grid point scale. The grid C# class can be constructed by itself in other classes. Alternatively, grids can be generated in the editor by using the monobehaviour wrapper. When using the monobehaviour wrapper, grids can be used with the object spawner which can spawn objects on grid points. This is good for making maps quickly when each object spawned should be aligned to a grid point.

Getting Started

Grid2DMono vs Grid2D



Grid 2D Mono Singleton (Script)

General
Axes: XZ
Grid Dimensions: X 20, Z 20
Grid Point Scale: Width 1, Length 1

Grid Lines & Corners
Calculate Grid Lines & Corners: ✓
Customize Line Width: ✓
Custom Line Width: 0.2

Show Grid Lines in Playmode: ✓
Grid Line Color
Grid Line Y Offset: 0.01

Debug
Show Grid Lines In Editor: ✓
Grid Line Editor Color
Grid Line Editor Y Offset: 0.01

Show Grid Corners In Editor: ☐
Show Grid Indices In Editor: ☐

Grid Points
Grid Line Vertices
Grid Corners

This is how the inspector looks for the Grid2DMono script, it's the easiest way to create and customize grids. The only reasons to construct the Grid2D non-monobehaviour class is if you want to have more control and add the ability to modify and construct grids during runtime. The Grid2D class is nearly as feature rich as Grid2DMono, the only differences are that Grid2DMono supports the ObjectSpawner while the raw class does not. Grid2D additionally can have either a dynamic or static Origin (Transform or Vector3) while Grid2DMono is always bound to the transform of the object the MonoBehaviour is attached to.

Grid2DMonoSingleton is the exact same thing as Grid2DMono. It's just a singleton version of it so it can be accessed statically using Grid2DMonoSingleton.Instance. While there can only exist one Grid2DMonoSingleton in the scene there can exist other non-singleton grids in the scene alongside it.

Setting up the Grid2DMono

As previously mentioned, the Grid2DMono and it's singleton variant are the easiest ways to use the grid. To set it up, simply create a new empty GameObject in a scene and add either the Grid2DMono or the Grid2DMonoSingleton script to it. The position/rotation of the object will determine the local space of the grid. The object can be moved during runtime and lines/grid points will follow. The settings of the grid can be changed in the inspector and any change to grid altering settings will regenerate the grid automatically.

In the inspector you can also click a button to add an object spawner. Which allows you to spawn objects on the grid using a variety of different spawning modes. By default the object spawner will delete and respawn all its objects when any value is changed. To change this behaviour select a different spawning setting at the top of the object spawner's inspector. AtAwake will spawn all objects at awake, manual will add a button at the bottom of the inspector to manually determine when to spawn objects.

Setting up the Grid2D in another MonoBehaviour

For games where rapid destruction/construction of new grids at runtime it is in most cases easier to just use the Grid2D class directly instead of using the Grid2DMono. In that case, just create a variable of type Grid2D in a monobehaviour of your choice. Then construct it when you find it appropriate to do so using one of the three constructors depending on what you want the origin of the grid to be:

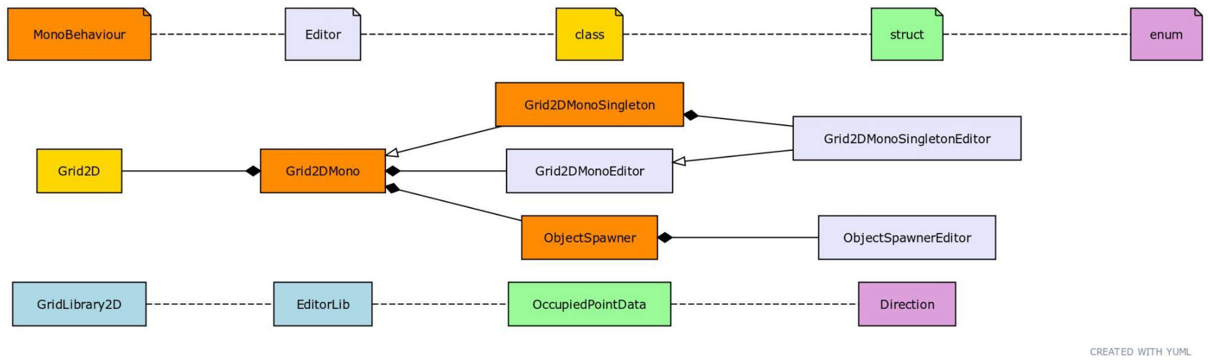
- If you want the origin to be a transform, just pass it a transform, the grid will now follow that transform and whenever you get world positions those will be updated based on the transform.
- If you want the origin to be constant, then pass in a Vector3 position and a Quaternion rotation. These can be changed by calling SetOriginOfGrid and shouldn't be super expensive, but in most cases it's easier to just use a transform for when you want the origin to update rapidly.
- If you want the origin to be 0,0,0 and rotation to be Quaternion.identity, pass in nothing.

Examples of usage

- **AI-pathfinding grid**: Let's say the game in question needs an AI pathfinding algorithm and the AI should have grid-based movement. You can simply create an empty in the scene and add the Grid2DMono or the singleton version of it. Then you specify size, scale and move it to where you want it (the same could be done with the Grid2D raw class if you wish also). Then you store the heuristic value (for A*) in a dictionary with the grid point index as a key. Then you can use utility methods in the Grid2D (which are also extended to the Grid2DMono), such as GetAdjacentIndex which will get the grid point index adjacent to the input index, given a direction.

- **Getting world positions on grid points**: For games that are aligned to a grid in some way you can use the grid to get the world positions of a grid point. The origin of the grid can be set to a constant position and rotation, or a transform (if the transform is moved during runtime you can get the grid points new world position (done internally using matrices)).
- **Map tool/generator for grid-based games**: For games with grid-based movement and/or games with art assets being aligned to a grid the ObjectSpawner can be used to make or generate maps. Let’s say we want to make a pacman-styled labyrinth quickly. Just create a Grid2DMono, specify size etc. And then add an ObjectSpawner, first create an object type set to spawning mode “All Indices”. Choose a cube and set the offset scale to x: 1, y: 0.1, z: 1. Then set snapping mode to “Mesh Bounds Top To Grid”. This will create a floor, then create a new object group, also cubes but set to specified indices or specified grid coordinates. Then turn on visual editing mode, you can now draw walls for the labyrinth on the grid (turn on “show grid lines in editor” in the Grid2DMono to be able to visualize the grid). You can alternatively do simple procedural generations with RandomIndices. Set the spawn setting to “AtAwake” for the grid to generate every time the game is played.
- **Visual grid**: You can also show any generated grids in the game itself by showing grid lines in playmode. With Grid2DMono, you just check the “Show Grid Lines in Playmode” bool. Now the grid lines will be rendered in the game during runtime. If you are using the raw Grid2D class, you simply call DrawGridLines in the unity event function “OnRenderObject).

Documentation



This is the general structure of the customizable game grid asset. Each class is well documented with in-line documentation and comments as well as regions. For examples of usage of this tool, refer to the examples folder. Here is a short summary of what each class is responsible for:

List of core classes

Grid2D - C# class that calculates 2D grids.
Grid2DMono - Monobehaviour wrapper for Grid2D class.
Grid2DMonoSingleton - Singleton version of Grid2DMono (inherits from it).
ObjectSpawner - Monobehaviour that can spawn objects on Grid2DMono's.

List of internal classes

Grid2DMonoEditor - Custom inspector code for Grid2DMono.
Grid2DMonoSingletonEditor - Custom inspector code for Grid2DMonoSingleton.
ObjectSpawnerEditor - Editor script for ObjectSpawner (inspector and visual editing HUD).
GridLibrary2D - Method library for simplifying code in Grid2D
OccupiedPointData - Struct with an object reference and the index of the point it occupies.
Direction - Enum that defines directions such as PositiveX, NegativeX etc.
EditorLib - Method library for simplifying custom inspector code.

List of files (in scripts folder)

Grid2D.cs, contains Grid2D
Grid2DMono.cs, contains Grid2DMono
Grid2DMonoSingleton.cs, contains Grid2DMonoSingleton
ObjectSpawner.cs, contains ObjectSpawner
Lib/GridLibrary.cs, contains OccupiedPointData, Direction, GridLibrary2D
Editor/Grid2DMonoEditor.cs, contains Grid2DMonoEditor
Editor/Grid2DMonoSingletonEditor.cs, contains Grid2DMonoSingletonEditor
Editor/ObjectSpawnerEditor.cs, contains ObjectSpawnerEditor
Editor/Lib/EditorLib.cs, contains EditorLib

List of utility methods in Grid2D/Grid2DMono

Only in Grid2D

Grid2D (constructor) - Takes a transform or a Vector3 pos and a Quaternion rotation to construct the grid and set its origin.
Generate - Takes grid values (such as axes, width, length, scale, etc.) and generates the grid based on those values.
SetOriginOfGrid - Sets the origin of the grid based on either a transform or a Vector3 pos and a Quaternion rotation.
SetOriginOfGridIfNull - Same as SetOriginOfGrid but only sets it if the grid doesn't currently have an origin of that type.
CalculateGrid - Generates the grid based on stored values (takes no arguments).
CalculateGridLines - Calculates all the grid line vertices (either 2 per line for non custom width or 4 per line for custom width. (This will automatically be called by CalculateGrid if the calculateGridLines bool is true).
DrawGridLines - Will draw grid lines using UnityEngine.GL (this should be called in OnRenderObject). Takes a color and a depth offset (how far above/below the grid the lines will be drawn).

DrawGridLinesInEditor - Will draw grid lines using gizmos (should be called in OnDrawGizmos). Takes a color and a depth offset (how far above/below the grid the lines will be drawn).
DrawGridCornersInEditor - Will draw spheres on the grid corners using gizmos (should be called in OnDrawGizmos). Takes a color and a radius for the spheres.
DrawGridIndices (Not compiled in build) - Draws the grid index number of each grid point on each grid point in 3d space using Handles.

Both in Grid2D and Grid2DMono

GetAdjacentIndex - Takes the index of a specified grid point and returns the adjacent grid point index based on the direction specified (can either be +/- x/y/z or +/- width/length).
GetWorldPointFromIndex - Takes the index of a specified grid point and returns the world position of it based on the localToWorld matrix of the origin of the grid.
GetGridPointsWorld - Gets an array of all grid points world space position (with each grid index matching up to the array index).
GetClosestIndex - Returns the index of the grid point that is closest to the given world space point.
IsPointOccupied - Checks to see if the given grid point index is marked as occupied, and if so returns the OccupiedPointData which contains a C# object reference to whatever is occupying that point.
SetPointOccupied - Marks the grid point (with the given index) as occupied given a C# object reference to the occupier.
ResetPointOccupied - Given the object reference that has previously been marked as occupying a point it will set that point as not occupied.
CalculateObjectRotation - Calculates the rotation an object should have if the up vector of the object should be parallel to the depth axis vector of the current grid (this rotation is in grid space, meaning that if the grid has been rotated the rotation returned will still be the same, to get the world space rotation perform this calculation: (gridRotation * theReturnedRotation)).
GetMiddleIndex - Will return the grid point index of the middle grid point (for even width/length, it will return the top right middle when looking straight at the grid with index 0 at the bottom left).
GridPointIndexToGridCoord - Returns a grid point coordinate (given a grid point index) originating from (0, 0) (width, length) at index 0 (in the bottom left of the grid of the specified grid point index), will return new Vector2Int(-1, -1) if index is out of bounds of the grid point array.
GridCoordToGridPointIndex - Returns the grid point index given a grid point coordinate originating from (0, 0) (width, length) at index 0 (in the bottom left of the grid of the specified grid point index), will return -1 if index is out of bounds of the grid point array.

Definitions

Axes - The specified axes of where the grid will generate. While the grid can be easily rotated, specifying the axes means that you can use xyz or the width/length/depth components to get adjacent grid points in a way that makes sense. Rotating will not change which grid point is returned when getting the adjacent using a specified direction while changing the axes will.

Grid Point Index - The index of the grid point (in the grid point array in Grid2D class). Index 0 is defined as the bottom left corner of the grid (if you are looking with the specified axes extending up and right (in screen space)). Indices increase in the horizontal direction and then vertical. The grid point indices can also be visualized with Grid2DMono by checking the “Show Grid Indices In Editor” bool.

Grid Coordinates - The coordinate of the grid point (can be converted to and from a grid point index with utility methods in Grid2D (forwarded from Grid2DMono). Coordiante 0,0 is in the bottom left corner of the grid (if you are looking with the specified axes extending up and right (in screen space)). The grid coordinates x axis is horizontal and the y axis is vertical.

Width - A specific axis of the grid depending on the selected axes. For XZ, width means X, For XY, width means X, For ZY, width means Z.

Length - A specific axis of the grid depending on the selected axes. For XZ, length means Z, For XY, length means Y, For ZY, length means Y.

Depth - A specific axis of the grid depending on the selected axes. For XZ, depth means Y, For XY, depth means Z, For ZY, depth means X.

Contact

If you have more questions, bug reports, feature requests or other thoughts you’d like to share, please contact me on mail@oliverlebert.com.