

Hypertonic Games- Grid Placement System

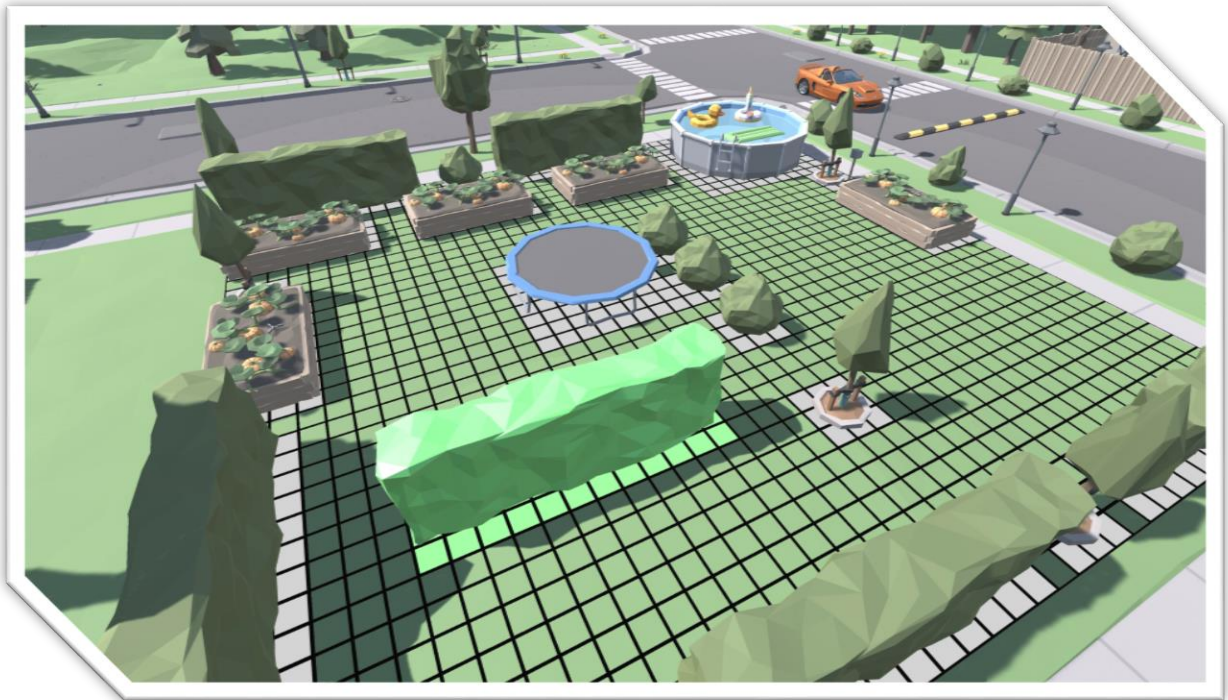


Table Of Contents

Table Of Contents	2
Overview	4
Compatibility.....	4
Getting Started	5
Installation	5
Quick Start	6
Grid Settings.....	10
Cell Image.....	10
Occupied Cell Image	10
Cell Colour Available	10
Cell Colour Not Available	10
Cell Colour Placed	10
Hide Placement Cells If Outside of Grid.....	10
Show Occupied Cells.....	11
Grid Canvas Camera Name	11
Change Object Materials	11
Object Placeable Material.....	11
Object Unplaceable Material.....	11
Default Alignment.....	11
Platform Input Type Mappings	11
Prevent Input Through UI	11
Width to Height Ratio	12
Grid Size	14
Horizontal Cell Count.....	14
Vertical Vell Count	14
Grid Position	14
Core Features.....	15
Placement mode.....	15
Move	15
Rotation	15
Change Alignment.....	16

Cancel Placement.....	17
Delete Object	17
Modifying the placement of an object.	18
Remove object	18
Clear Grid	18
Saving and Loading	19
Add Programmatically	20
Grid Manager Accessor	21
Multiple Grids	22
Mobile Support	22
Supports 100 million Grid Cells.....	22
Upcoming features	24
Support	24

Overview

The grid placement system was developed to allow developers to quickly and easily drop a grid placement system into their Unity project. The asset is specifically targeted for allowing players to place and modify objects onto a grid at runtime.

Compatibility

The supported unity version for this asset is 2019 LTS or later.

Getting Started

Installation

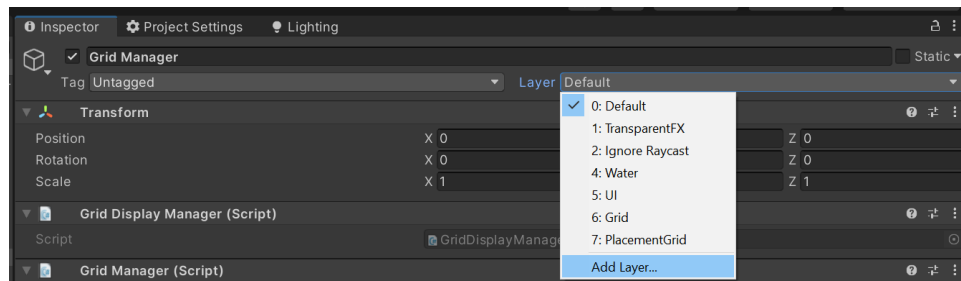
1. Open your project in Unity.
2. Import the .unitypackage file into the project.

Important

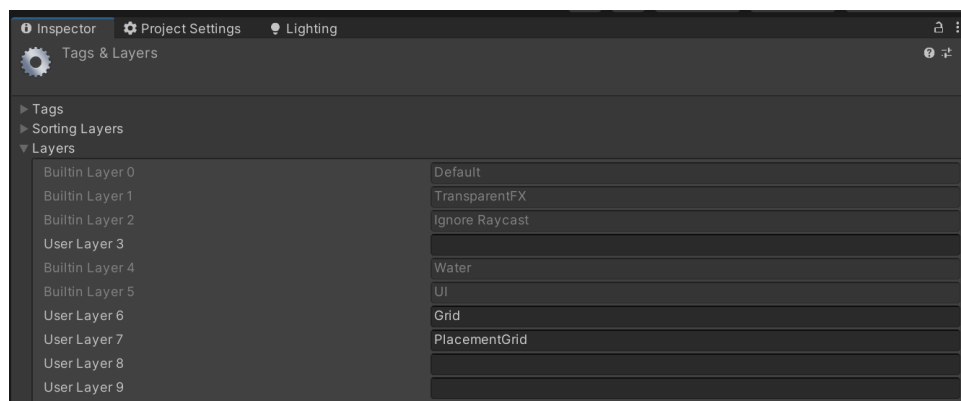
If using **Unity 2019** the creation of the required Unity layers for the grid placement system does not happen automatically. So you will need to create the layers manually in the Unity Editor. The names of the layers are:

- Grid
- PlacementGrid

To add new layers in your project, select the Layer dropdown within the inspector tab. Select Add Layer...



Then in the next empty space after the built-in layers input the layer names: **Grid** and **PlacementGrid**

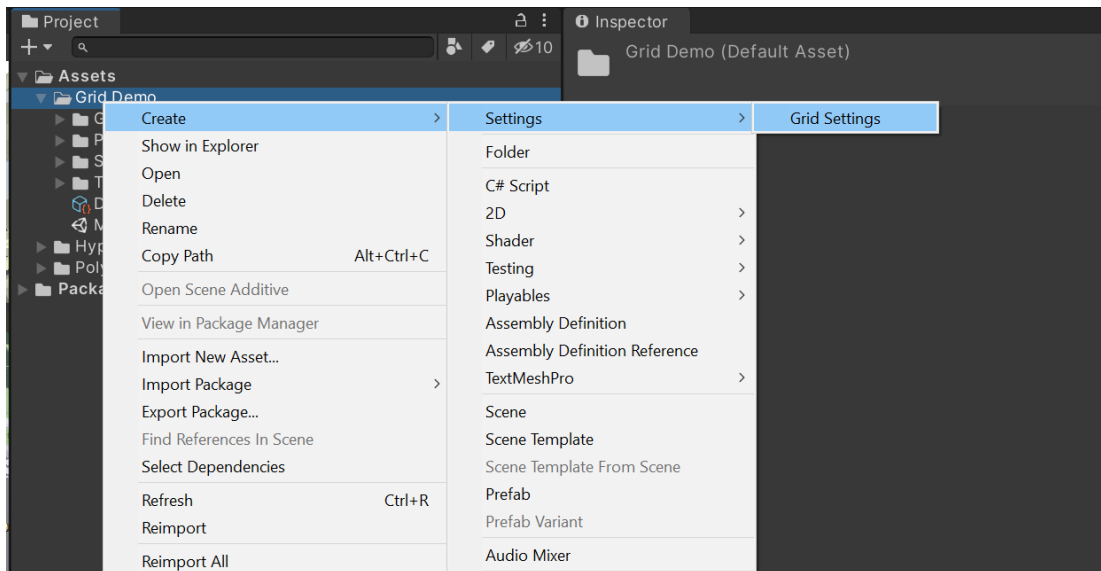


Quick Start

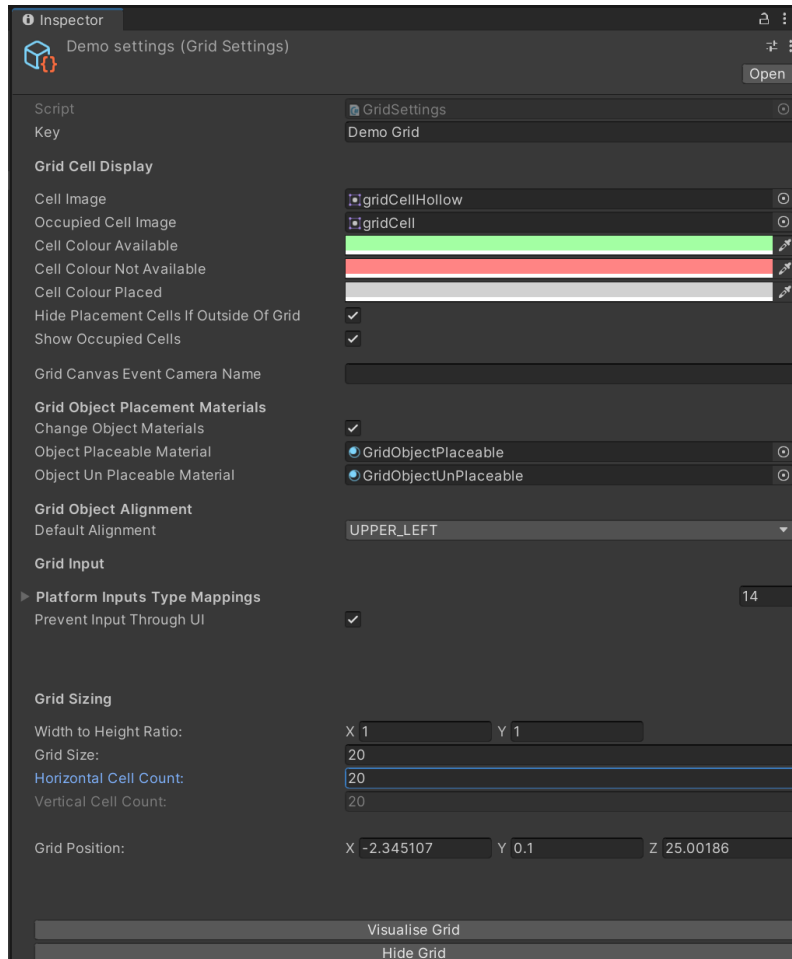
The Basic Grid Demo example scene will best demonstrate how to interface and interact with the Grid System.

To implement the grid placement system into a scene follow these steps:

1. Create a new GridSettings asset in your Project folder. You can add a settings object by right clicking and selecting: Create > Settings > Grid Settings



2. Populate the settings object. (See the Grid Settings section for more details).



3. Create a Grid Manager. You can either drop the GridManager component onto a game object or create the component via code.
4. In another script you'll need a reference to the GridManager component and the settings object. Once you have those 2 references call the Setup function on the GridManager and pass in the Grid Settings.

```

1  private void CreateGridManager()
2  {
3      GameObject gridManagerObject = new GameObject("Grid Manager");
4      GridManager gridManager = gridManagerObject.AddComponent<GridManager>();
5      gridManager.Setup(_gridSettings);
6  }

```

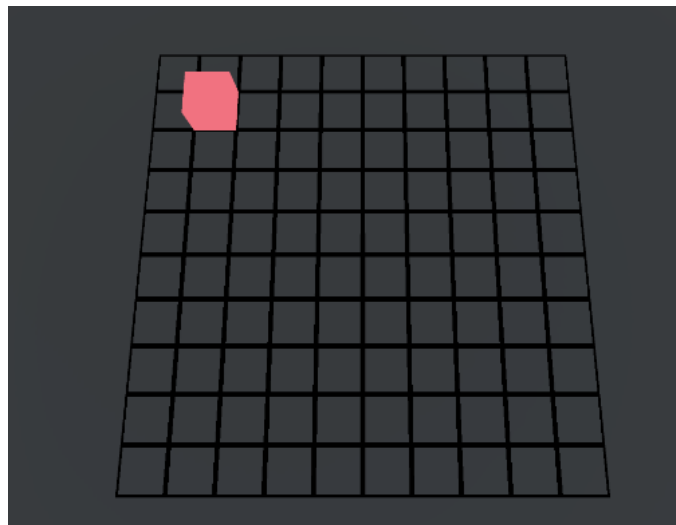
5. Pass an object to the Grid Manager to place. If you don't want to store a reference to the GridManager in the class you can use the **GridManagerAccessor** to access it instead (See the Grid Manager Accessor section for more details).

```

1  GameObject objectToPlace = Instantiate(_gridObjectToSpawnPrefab, GridManagerAccessor.GridManager.GetGridPosition(), new Quaternion());
2  GridManagerAccessor.GridManager.EnterPlacementMode(objectToPlace);

```

6. Move the object into the desired position on the grid. Once in placement mode you can manipulate the object in several ways, (see the Placement Mode section for more details).



7. Confirm the placement. The confirm placement function will return a bool to inform you if the object was able to be placed or not. If it's an invalid placement it may be due to the fact the object is overlapping another object placed on the grid, or it may be partly off the grid.



```
1 bool placed = GridManagerAccessor.GridManager.ConfirmPlacement();
```

Configuring Objects for the Grid

The grid system is designed to not require objects to have any configuration done before being passed to the grid system. When an object is passed to the grid system it'll calculate the size of the object and how many grid cells that object occupies based on the colliders.

The grid placement system will recursively search each child object of the object and calculate the size required based on each collider on the parent and child objects.

Grid Settings

The grid is generated using a settings object that allows the user to define many aspects of the grid and how the placement of an object behaves.

Cell Image

The grid settings take in a Unity sprite to use as the visualisation of the grid. The grid is generated by repeating this sprite to create the grid cells. The sample scenes contain examples of a simple square and circle grid cell.

Occupied Cell Image

This sprite is used to display the currently unavailable grid cells due to an object occupying those cells. It's also used to show the grid cells that the item being placed will occupy if placed at that position.

Cell Colour Available

The colour of the cells when the item is being placed is valid.

Cell Colour Not Available

The colour of the cells when the item is being placed is not valid.

Cell Colour Placed

The colour of the cells of occupied cells.

Hide Placement Cells If Outside of Grid

If part of the object being placed is outside the grid hide or show those cells.

Show Occupied Cells

Defines whether the cells that are occupied are displayed differently to the normal grid cells.

Grid Canvas Camera Name

The Grid Manager generates a UI Canvas to display the grid image. This Unity canvas can have an event camera assigned to it. To do this, provide the name of the game object that the desired camera is attached to. If left blank, Camera.Main is used.

Change Object Materials

When an object is being placed the materials of the object are swapped to either the **Object Placeable Material** or the **Object Unplaceable Material** depending on if the placement is valid. To prevent the object's material being changed while in placement mode, turn this setting off.

Object Placeable Material

The material applied to an object when the placement of the object is valid.

Object Unplaceable Material

The material applied to an object when the placement of the object is invalid/

Default Alignment

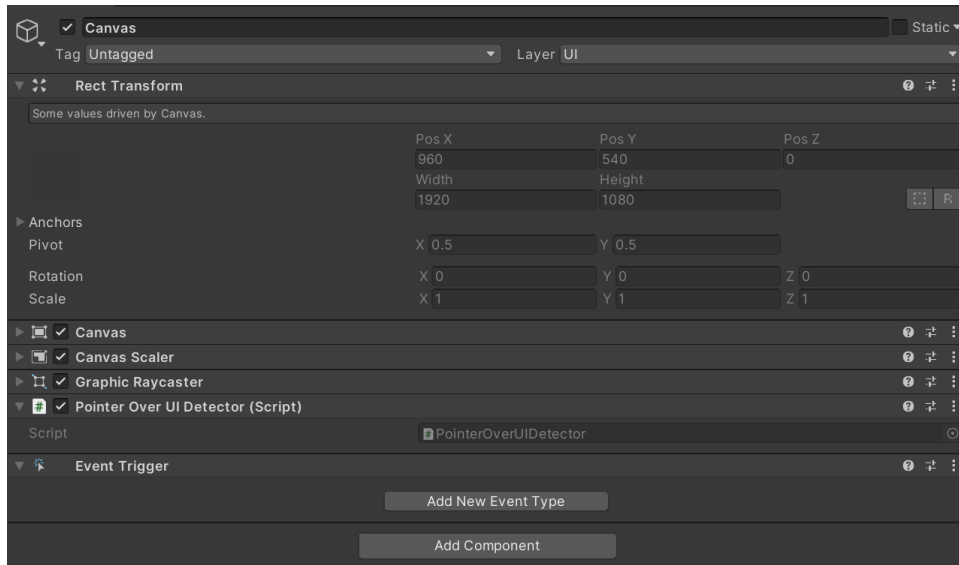
Defines the alignment of the grid object when being placed onto the grid. The alignment of the object can be changed when being placed. (See the alignment section in Placement Mode for more details).

Platform Input Type Mappings

Defines the input type for each Unity runtime platform. The current input types are composed of Touch and Mouse types.

Prevent Input Through UI

When this setting is on it'll prevent the raycast that detects interactions with the grid. This may be needed when you have a UI menu on top of the grid and don't want to be moving the selected grid object behind the menu. For this setting to work the GameObject containing the canvas of the UI will also need the "Pointer Over UI Detector" component attached. (See Basic Demo Scene for a working example).



Width to Height Ratio

This defines the ratio of the width to the ratio of the height of the grid. For any square grid this can be left as a 1:1 ratio. You can adjust the width to height ratio to create an elongated shaped grid. Because adjusting the ratio of width to height would stretch the grid cells in either the x or y dimension the vertical cell count must be updated to ensure each grid cell is still a square within a non-square grid.

To help with this, the vertical cell count is automatically updated based on the new ratio of width to height and the horizontal cell count. Although the vertical cell count is calculated for you, you'll still need to ensure that the dimensions are valid.

For example, look at the image below.

Grid Sizing

Width to Height Ratio:

X 1

Y 2

Grid Size:

100

Horizontal Cell Count:

10

Vertical Cell Count:

20

Grid Position:

X 0

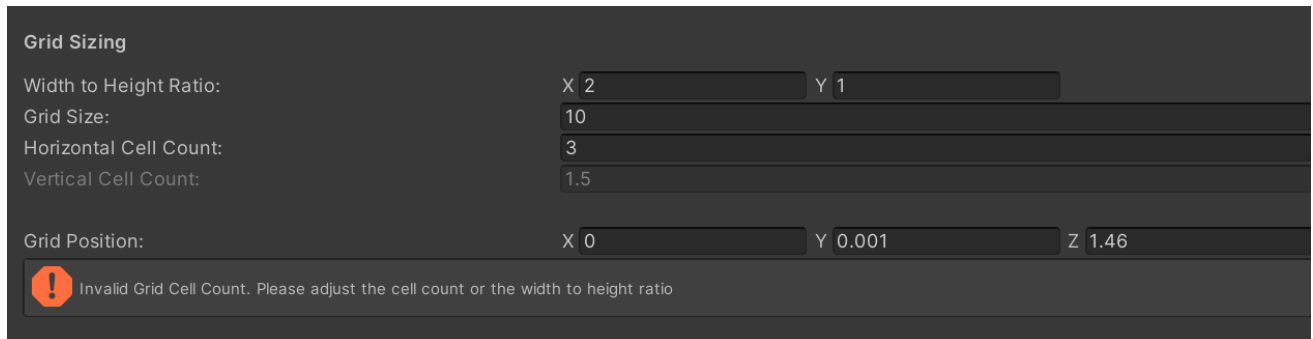
Y 0.001

Z 1.46

You can see the width to height ratio is 1:2. This means that because the grid size is 100, the width of the grid will be 100 and the height will be 200. As there is a horizontal cell count of 10, the vertical cell count is adjusted to 20 (matching the width to height ratio). If the vertical cell count remained at 10 then each grid cell would be twice tall as it is wide.

Important

When adjusting the ratio or horizontal cell count you must ensure the values are valid. As the vertical cell count is adjusted automatically it could result in a non-integer number. When this happens, an error will be shown in the inspector to inform you the values are invalid. If the error is ignored in the settings object, the grid manager will throw an error when setting up. See before for an example of invalid values.

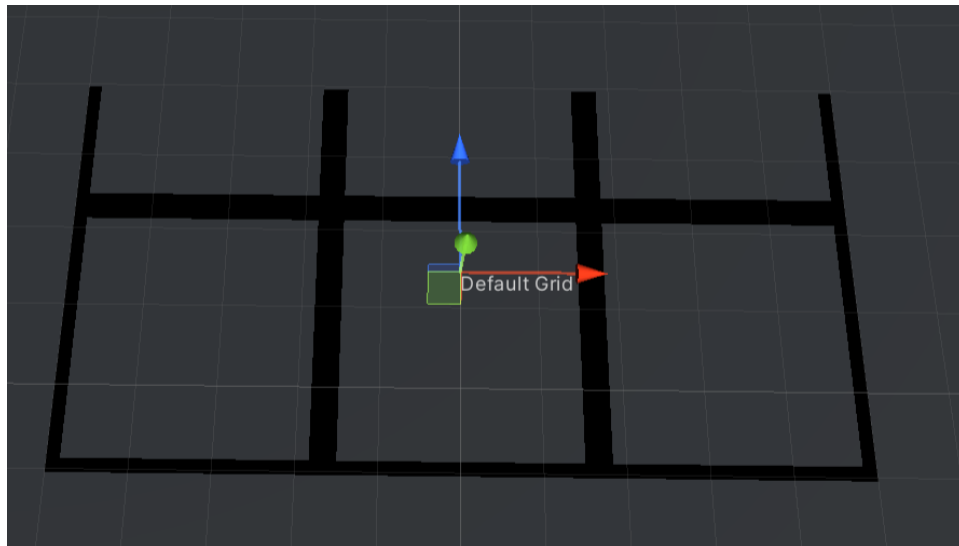


The screenshot shows a 'Grid Sizing' inspector panel with the following settings:

Property	Value
Width to Height Ratio:	X 2 Y 1
Grid Size:	10
Horizontal Cell Count:	3
Vertical Cell Count:	1.5
Grid Position:	X 0 Y 0.001 Z 1.46

Below the settings, an orange warning icon is displayed next to the text: "Invalid Grid Cell Count. Please adjust the cell count or the width to height ratio".

As you can see above the ratio is 2:1 and there are only 3 horizontal cells. The ratio of 2:1 == 0.5. $3 * 0.5 = 1.5$. As you cannot have half a cell the grid is invalid.



Grid Size

This determines the width and height of the grid in Unity world space. When the width to height ratio is adjusted the grid size will control the width and the height will be calculated based on the ratio.

Horizontal Cell Count

This determines how many cells the grid will contain in its x axis.

Vertical Vell Count

This property cannot be changed manually. Instead this value is driven by the width to height ratio setting in combination with the horizontal cell count. Please see the Width to Height property listed above for more details on this.

Grid Position

This sets the position of the grid in Unity world space. When the grid settings are selected you can use the transform handles in the scene view to move the grid into the desired location.

Core Features

Placement mode

Placement mode is where an object is passed to the Grid Manager to place onto the grid. Once in placement mode, a number of options are available.

Call **EnterPlacementMode** to add a new object to the grid. Once called the grid will be shown and the object will follow the mouse around the grid when held down.

A screenshot of a code editor window with a dark background and light blue borders. The window has three colored window control buttons (red, yellow, green) in the top-left corner. A single line of C# code is visible, starting with a line number '1' in light blue. The code is: `GridManagerAccessor.GridManager.EnterPlacementMode(objectToPlace);` The text is in a light blue monospace font.

```
1 GridManagerAccessor.GridManager.EnterPlacementMode(objectToPlace);
```

Important

If you want to have UI on top of the grid while the grid is being displayed you'll most likely not want to move the grid object on the grid if interacting with the UI. To block the raycasts from going through the UI when in placement mode see the **Prevent Input Through UI** section in the grid settings above.

Move

You can move the object around the grid by holding down the mouse button and moving the cursor to the desired location. (Touch support also supported).

Rotation

When the object is being placed on the grid the user may need to rotate the object to achieve the desired position. The object should be rotated independently using the normal Unity rotation techniques. It is the user's responsibility to handle the rotation so they can apply any animations or extra styling to the object when it rotates. However, once rotated the "HandleGridObjectRotated" function should be called to let the Grid Manager know it needs to recalculate if the new position of the object is valid.

```

1 private void HandleRotateRightPressed()
2 {
3     _selectedGridObject.transform.Rotate(new Vector3(0, 90, 0));
4
5     GridManagerAccessor.GridManager.HandleGridObjectRotated();
6 }

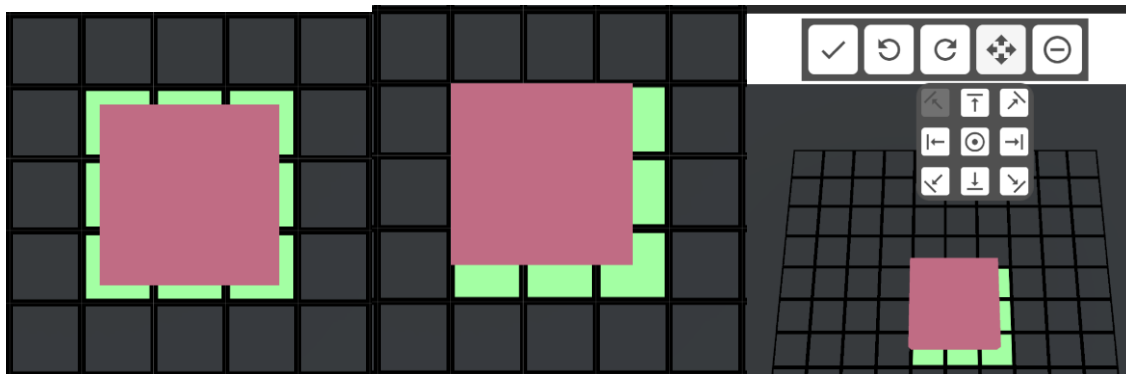
```

Important

For the grid placement system to work correctly with object rotations. The object must be rotated in increments of 90 degrees on the y axis.

Change Alignment

If the object being placed does not fit perfectly within the grid cells it occupies it will have padding. Changing the alignment of the object will change which edge(s) of the occupied cells the object is snapped to. You can change the alignment of the object so that the object is snug against any of the edges or corners of the cells the object occupies. It can also be aligned to the center of the cells it occupies to create even padding surrounding the object.



The default alignment is set in the grid setting object. However, to change the alignment when in placement mode call the “ChangeAlignment” function on the Grid Manager.


```
1 private void HandleChangeAlignmentPressed(ObjectAlignment objectAlignment)
2 {
3     GridManagerAccessor.GridManager.ChangeAlignment(objectAlignment);
4 }
```

Cancel Placement

If the object is in placement mode and has not already been placed, then you can cancel the current placement. This will destroy the object currently being placed and hide the grid. To cancel the placement of an object call the “CancelPlacement()” function on the Grid Manager.

```
1 private void HandleRotateRightPressed()
2 {
3     _selectedGridObject.transform.Rotate(new Vector3(0, 90, 0));
4
5     GridManagerAccessor.GridManager.HandleGridObjectRotated();
6 }
```

Delete Object

If you wish to delete an object that has already been placed on the grid you can achieve this by calling the DeleteObject function on the Grid Manager and pass in the object to delete.

```
1 private void HandleDeleteObjectPressed()
2 {
3     GridManagerAccessor.GridManager.DeleteObject(_selectedGridObject);
4     _selectedGridObject = null;
5 }
```

Modifying the placement of an object.

If an object has already been placed on the grid, you can re-enter placement mode to achieve the desired placement. To do this, call “ModifyPlacementOfGridObject”.

```
1 private void HandleExampleGridObjectSelected(GameObject gridObject)
2 {
3     GridManagerAccessor.GridManager.ModifyPlacementOfGridObject(gridObject);
4 }
```

Remove object

To remove all references to an object and convert the occupied cells back to unoccupied without destroying the grid object you can call “RemoveObjectFromGrid”.

This is useful for scenarios where the user wishes to place an object at a certain grid position that once placed will move on its own. (Essentially using the grid placement system to snap the game object to the desired grid location to spawn an object in a controlled fashion).

```
1 private void HandleRemoveObjectPressed()
2 {
3     GridManagerAccessor.GridManager.RemoveObjectFromGrid(_selectedGridObject);
4     _selectedGridObject = null;
5 }
```

Clear Grid

To delete all the current items in the grid call the “ClearGrid(bool destroyGridObjects = true)” function on the Grid Manager. To just remove all references of the objects from the grid without deleting the objects you can pass in false as a parameter.

```
1 private void DeleteAllGridObjects()
2 {
3     GridManagerAccessor.GridManager.ClearGrid();
4 }
```

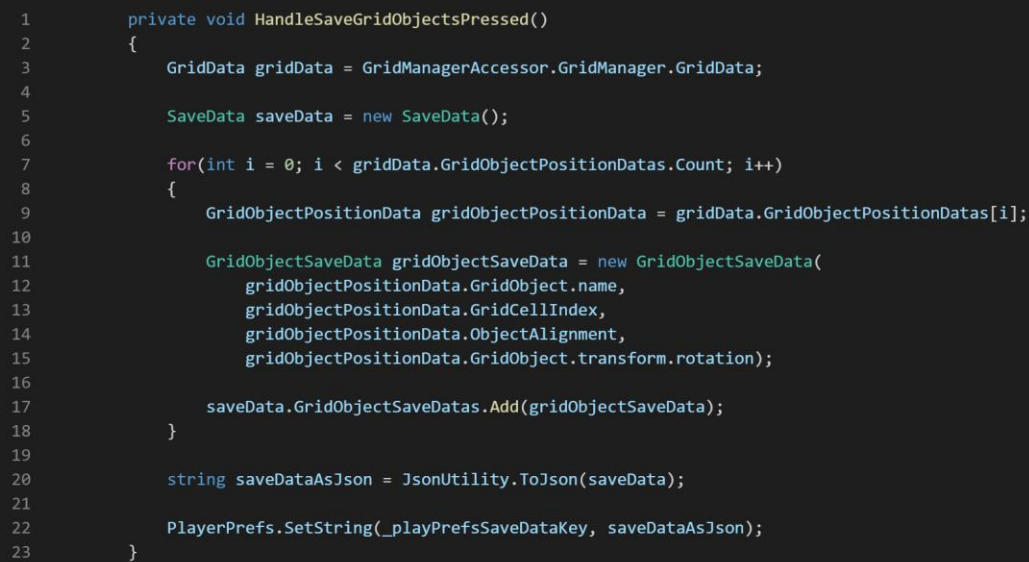
Saving and Loading

There may be scenarios where the objects in the grid need to be persisted between levels/sessions. The Grid Manager offers functionality to easily meet these requirements. (See the “Grid Save Manager” class in the sample scene for an example of how this may be implemented.

The Grid Manager exposes two functions to achieve the desired functionality.

Accessing the Grid Data

The current grid data can be accessed via the “GridData” Property on the Grid Manager.



```
1 private void HandleSaveGridObjectsPressed()
2 {
3     GridData gridData = GridManagerAccessor.GridManager.GridData;
4
5     SaveData saveData = new SaveData();
6
7     for(int i = 0; i < gridData.GridObjectPositionDatas.Count; i++)
8     {
9         GridObjectPositionData gridObjectPositionData = gridData.GridObjectPositionDatas[i];
10
11         GridObjectSaveData gridObjectSaveData = new GridObjectSaveData(
12             gridObjectPositionData.GridObject.name,
13             gridObjectPositionData.GridCellIndex,
14             gridObjectPositionData.ObjectAlignment,
15             gridObjectPositionData.GridObject.transform.rotation);
16
17         saveData.GridObjectSaveDatas.Add(gridObjectSaveData);
18     }
19
20     string saveDataAsJson = JsonUtility.ToJson(saveData);
21
22     PlayerPrefs.SetString(_playPrefsSaveDataKey, saveDataAsJson);
23 }
```

Populating with Grid Data

Once you have retrieved the grid data that has been persisted between sessions you can load it into the grid. To populate the grid with the grid data object pass it to the “PopulateWithGridData” function on. This is an async method and may need to be awaited in certain scenarios.

```

1 private async Task LoadGridData()
2 {
3     if (!PlayerPrefs.HasKey(_playPrefsSaveDataKey))
4     {
5         Debug.LogWarning("There is no save data stored yet. You must save the grid data before being able to load it.");
6         return;
7     }
8
9     string saveDataAsJson = PlayerPrefs.GetString(_playPrefsSaveDataKey);
10
11     SaveData saveData = JsonUtility.FromJson<SaveData>(saveDataAsJson);
12
13     List<GridObjectPositionData> gridObjectPositionDatas = new List<GridObjectPositionData>();
14
15     foreach (GridObjectSaveData gridObjectSaveData in saveData.GridObjectSaveDatas)
16     {
17         GameObject prefab = _gridObjectPrefabs.Where(x => x.name.Equals(gridObjectSaveData.PrefabName))
18             .FirstOrDefault();
19         GameObject gridObject = Instantiate(prefab);
20         gridObject.transform.rotation = gridObjectSaveData.ObjectRotation;
21
22         // Remove the "(Clone)" from instantiated name.
23         gridObject.name = prefab.name;
24
25
26         if (!gridObject.TryGetComponent(out ExampleGridObject exampleGridObjectComponent))
27         {
28             gridObject.AddComponent<ExampleGridObject>();
29         }
30
31         GridObjectPositionData gridObjectPositionData = new GridObjectPositionData(
32             gridObject,
33             gridObjectSaveData.GridCellIndex,
34             gridObjectSaveData.ObjectAlignment);
35
36         gridObjectPositionDatas.Add(gridObjectPositionData);
37     }
38
39     GridData gridData = new GridData(gridObjectPositionDatas);
40
41     await GridManagerAccessor.GridManager.PopulateWithGridData(gridData, true);
42 }

```

There is an optional parameter that provides the option to clear the grid of its current objects prior to populating it with the grid data. It is recommended to clear the grid prior to populating the data as it will mitigate the risk of invalid placements due to overlapping objects. However, there may be use cases where the grid needs to have the grid data combine with the existing grid objects.

Add Programmatically

As well as adding objects to the Grid in bulk with the “PopulateWithGridData” function mentioned above. There is also the option to add a single object to the grid without entering placement mode.

To achieve this call “AddObjectToGrid” on the Grid Manager. You will need to pass in the object to place on the grid, the grid cell index to place the object at, and the desired alignment of the object.

```
1  await GridManagerAccessor.GridManager.AddObjectToGrid(  
2      gridObject,  
3      gridObjectPositionData.GridCellIndex,  
4      gridObjectPositionData.ObjectAlignment);
```

Important

If you need to call `AddObjectToGrid` multiple times in short succession, E.G., in a loop. Each call must be awaited.

Grid Manager Accessor

The Grid Manager Accessor provides an effortless way to access the Grid Manager class without the need for storing a reference to it. Call `GridManagerAccessor.GridManager` to access the default Grid Manager.

```
1  GridManagerAccessor.GridManager
```

The Grid Manager class will register itself to the Grid Manager Accessor when it's set up. The key used to register itself is the key in the Grid settings object provided to the Grid Manager when it's set up. If you have multiple Grid Managers, you can access them individually through the Grid Manager Accessor class by calling `GridManagerAccessor.GetGridManagerByKey` and passing in the Grid Manager's Key.

```
1  GridManagerAccessor.GetGridManagerByKey("MY_GRID_KEY");
```

If switching between two Grid Manager's frequently you can set the selected Grid Manager in the class by calling `SetSelectedGridManager`. This means that when you call `GridManagerAccessor.GridManager` the selected Grid Manager will be returned.

```
1  GridManagerAccessor.SetSelectedGridManager("MY_PRIMARY_GRID_KEY");
```

This class has been created to provide an easy way to reference the Grid Manager. However, there is no reason you cannot reference the Grid Manager directly in a script if you choose.

Important

If you have multiple Grid Managers, you must ensure that the key set in the grid settings object is unique.

Multiple Grids

This asset supports multiple Grid Managers. If you use the Grid Manager Accessor class to obtain a reference for the Grid Manager, you can specify which Grid Manager you wish to access. See the “Grid Manager Accessor” section above to learn how you can access different Grid Managers.

Mobile Support

The Grid Placement System supports mobile touch controls on the grid. By default the Android and iOS runtime platforms set the Input Type to Touch controls. To change this or configure more runtime platforms modify the settings in the Platform Input Type Mappings sections of the grid settings.

Supports 100 million Grid Cells

The grid can support millions of grid cells within a single grid. The system has been tested and can generate a grid with 100 million cells. Such a large grid takes time to generate.

However, it’s ultimately up to the hardware of the device. The grid is designed to be very performant when generating the grid when “Setup” is called on the Grid Manager.

Grids with a cell count of 100,000 generate in less than 1 second (on an 11th Gen i7 processor). As the grid cell count increases the grid will take longer to generate. With a grid cell of 10,000,000 it takes ~1 minute (on an 11th Gen i7 processor). For a grid cell count of 100,000,000, it takes ~10 minutes on an 11th Gen i7 processor), however it consumes a lot of memory so is not recommended. The grid cell generation is threaded so it does not block the game.

Important

For **extremely large** cell counts the grid will take time to generate when it’s set up for the first time. (Once set up the grid will show and hide instantly). As the grid set up is threaded to avoid blocking the main thread you need to ensure that the grid has finished generating before trying to place an object on the grid. When grids are so large that they are not generated instantly you can wait for a setup callback

to be invoked to ensure the grid is ready to use. (See the large grid cell demo scene which demonstrates using this callback to wait for a grid to generate that contains 10 million cells.

Upcoming features

This is currently version 1.0.0 of the system. This asset will continue to be developed and improved. If you have any feature requests, please feel free to submit your requests to the support email.

Upcoming feature list:

- Place multiple items at once.
- Delete multiple items at once.
- Add a custom size for each object that'll override the size calculated based on the colliders.
- Update the Grid Position and all its objects at runtime.

Support

If you experience any issues or need assistance with this asset, please send an email to support@hypertonicgames.com.