

## [Tilemap](#)

[Tile Asset](#)

[Creating Tiles](#)

[How to create a Tile asset](#)

[How to generate multiple Tile assets](#)

[How to create a Tilemap](#)

[Adjusting Grid for Tilemap](#)

[Tilemap Palette](#)

[Editing Tilemap Palette](#)

[Painting Tilemaps](#)

[Tilemap Focus mode](#)

[Physics 2D and Tilemaps](#)

[How to create a Scriptable Tile](#)

[TileBase](#)

[Tile](#)

[Other Useful Classes:](#)

[TileFlags](#)

[Tile.ColliderType](#)

[ITilemap](#)

[TileData](#)

[TileAnimationData](#)

[An example Scriptable Tile:](#)

[RoadTile](#)

[How to create a Scriptable Brush](#)

[GridBrushBase](#)

[GridBrushEditorBase](#)

[Other Useful Classes:](#)

[GridBrushBase.Tool](#)

[GridBrushBase.RotationDirection](#)

[GridBrushBase.FlipAxis](#)

[A Scriptable Brush example:](#)

[2D Extras in GitHub](#)

# Tilemap

## Tile Asset

Typically tiles are actual sprites that are arranged on a Tilemap. In Unity's implementation, we use an intermediary asset that references the sprite instead. This enables us to extend the tile itself in many ways, creating a robust and flexible system for tiles and Tilemaps.

### Properties



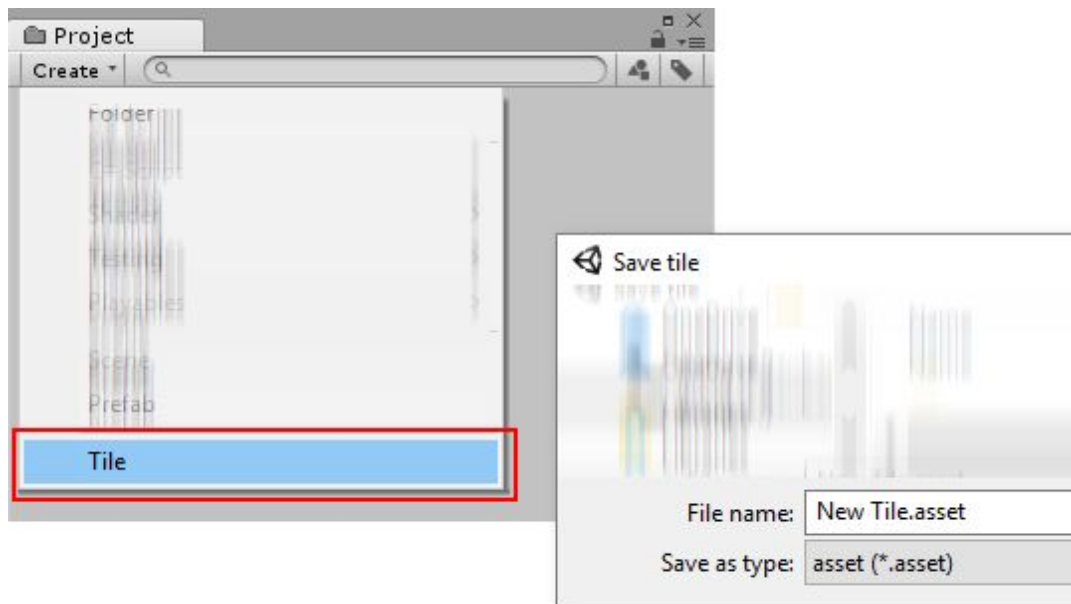
Property	Function
Sprite	The sprite that is used by this tile asset
Color	Used to tint the material
Collider Type	None, Sprite or Grid

## Creating Tiles

There are two ways to create Tiles. The first method is to directly create a Tile asset. The other method is automatically generate the Tiles from a selection of sprites.

### How to create a Tile asset

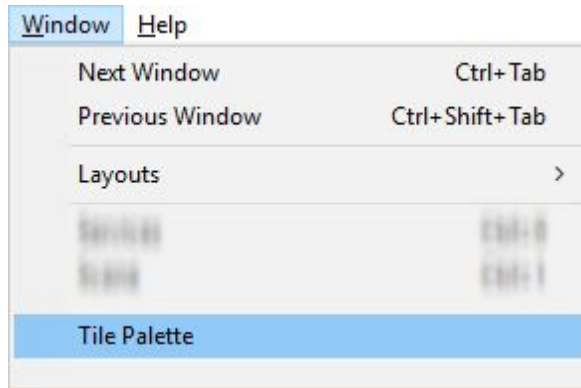
To create a Tile select from the Project menu Create > Tile. Then select where to save the new Tile asset.



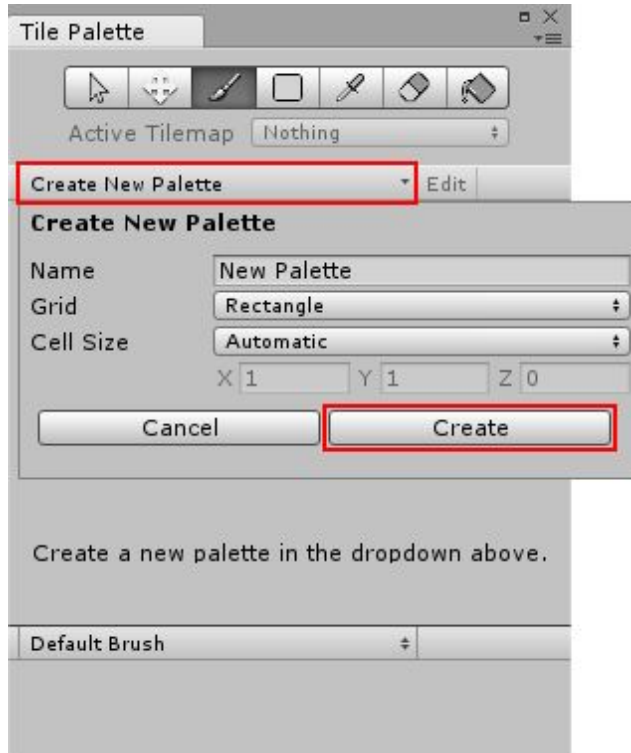
### How to generate multiple Tile assets

Automatic or multiple Tile creation requires a Palette to be loaded in the Tile Palette. If there is no Palette loaded you will have to create a new Palette.

To create a new Palette, open the Tile Palette from Window > Tile Palette.

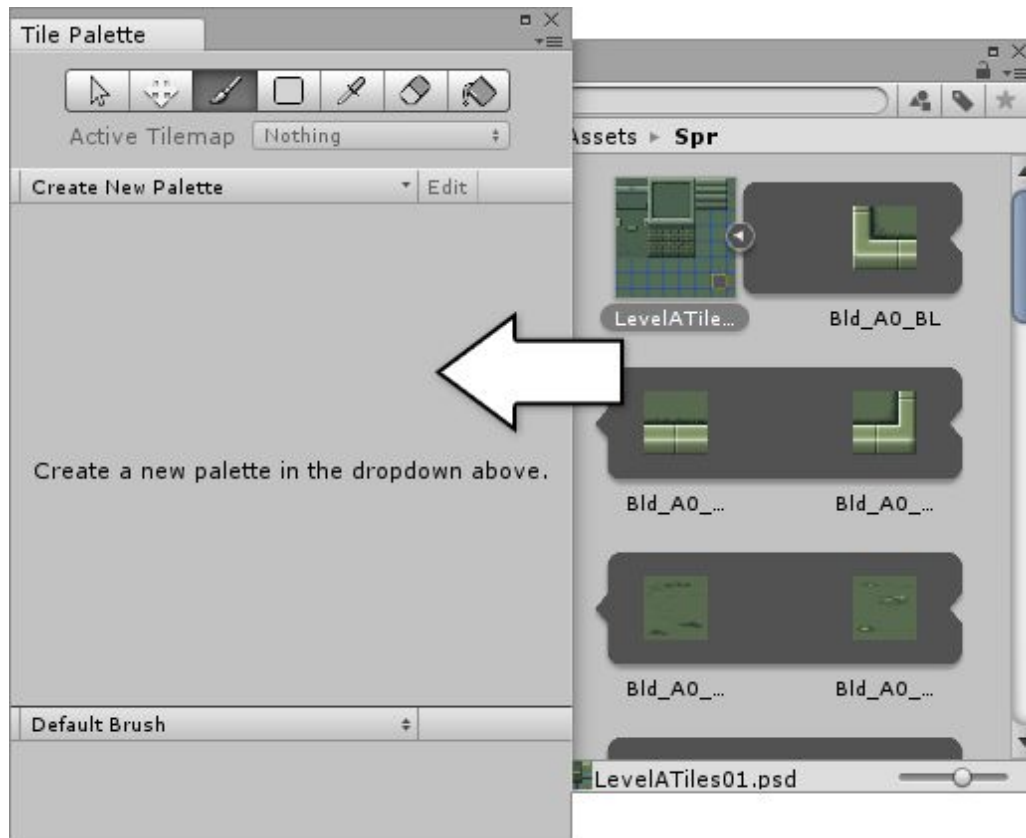


Click the Create New Palette button in the Tile Palette. Provide a name for the Palette and click the Create button. Select a folder to save the Palette. The created Palette will be automatically loaded.



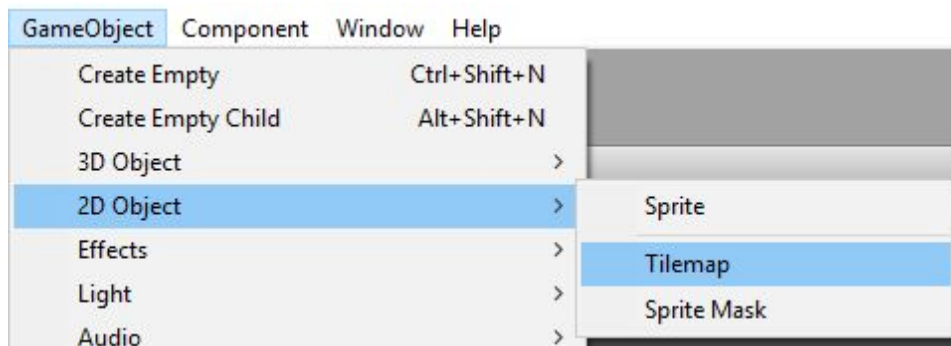
Drag and drop Textures or Sprite/s from the Assets folder onto the Tile Palette. Choose where to save the new Tile assets. New Tile assets will be generated in the selected folder and the

tiles will be placed on the Tile Palette. **Remember to save your project to save the Palette.**

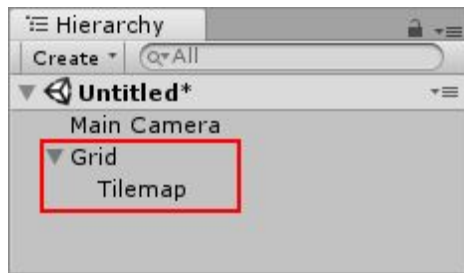


## How to create a Tilemap

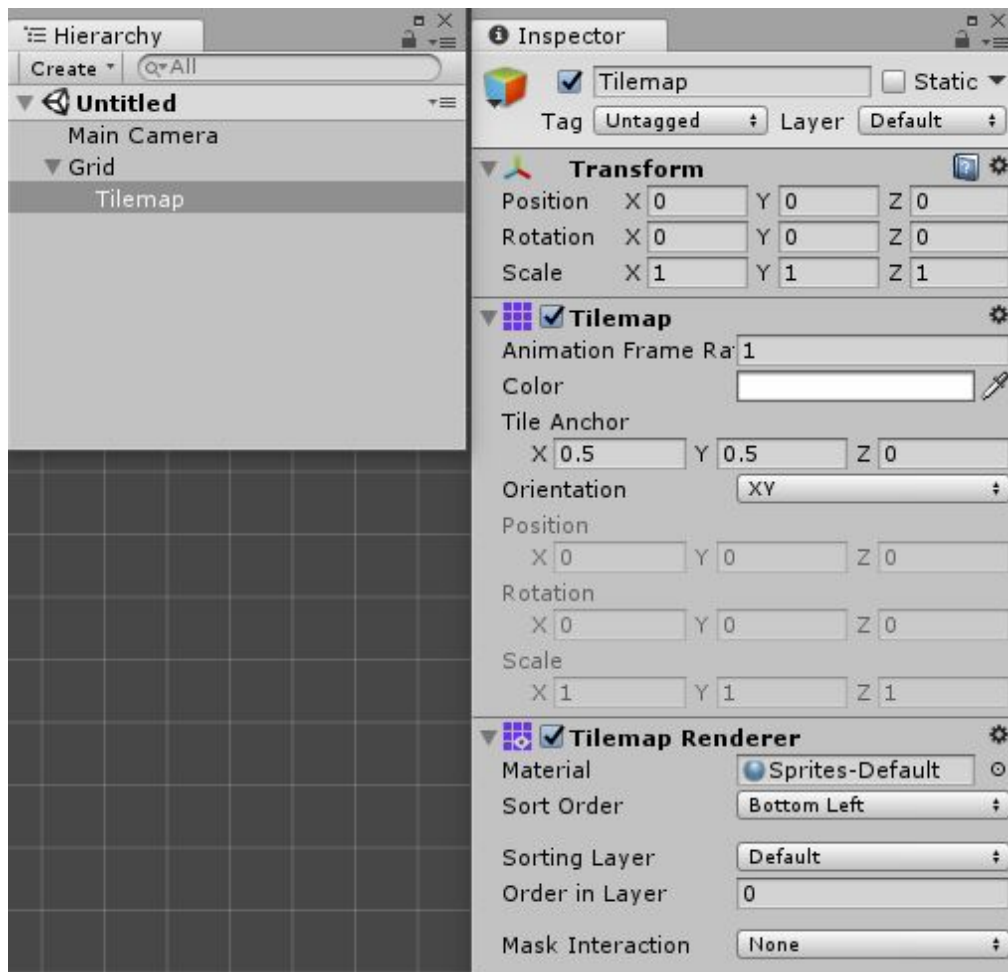
From the GameObject Menu, move to 2D Object and select Tilemap



This will create a new GameObject with a child GameObject in the scene. The GameObject is called Grid which determines the layout of child Tilemaps.

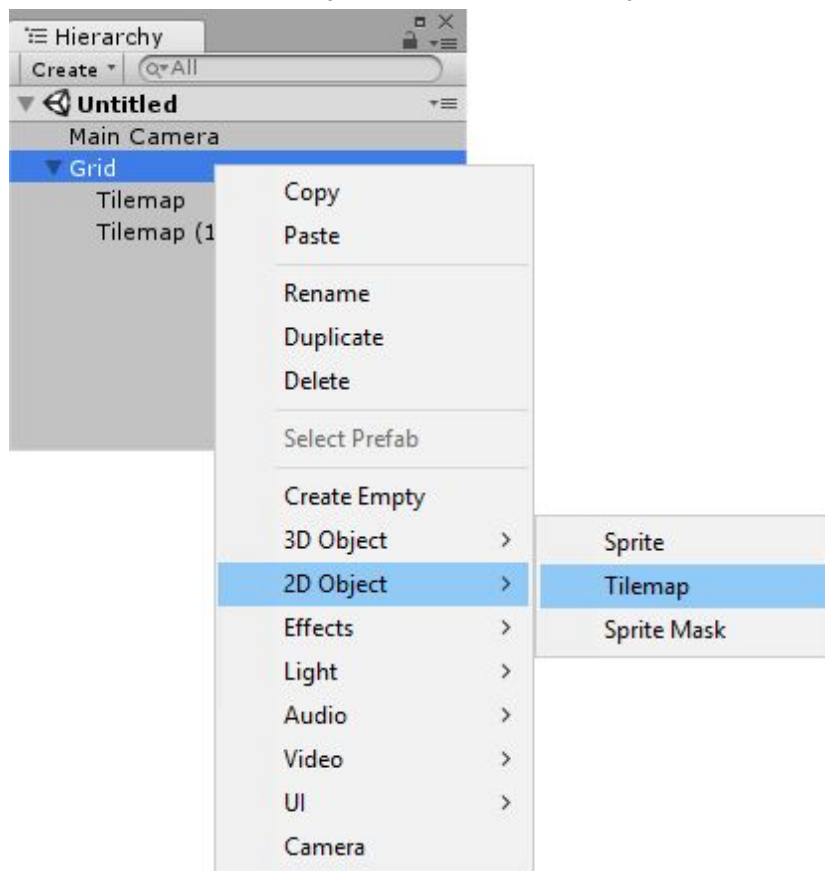


The child GameObject is called Tilemap which is comprised of a Tilemap Component and Tilemap Renderer Component. The Tilemap GameObject is where tiles are painted on.

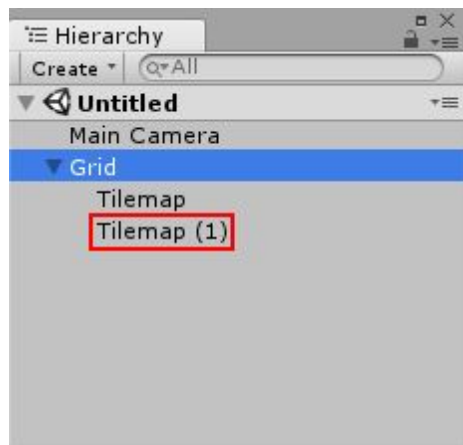


To create additional Tilemaps to be used as “layers”, select the Grid GameObject or the Tilemap GameObject, and select GameObject > 2D Object > Tilemaps in the menu or right-click

on the selected GameObject and click on 2D Object/Tilemap.



A new GameObject called Tilemap will be added into the hierarchy of the selected GameObject. You can paint tiles on this new GameObject as well.



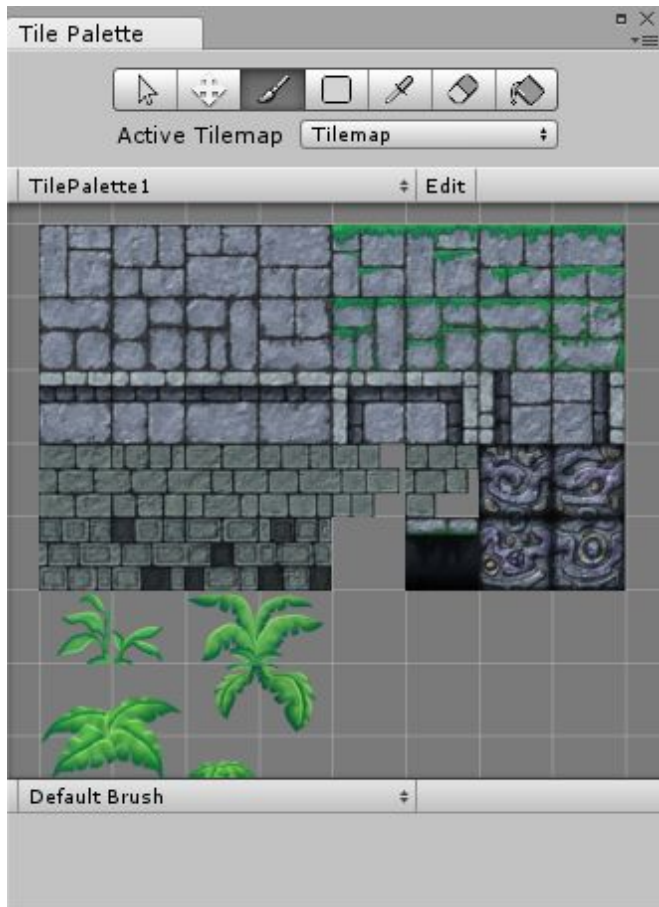
## Adjusting Grid for Tilemap



1. Select the Grid GameObject.
2. Adjust the values in the Grid component in the inspector.
  - a. Cell Size: Size of each cell in the Grid
  - b. Cell Gap: Size of the Gap between each cell in the Grid
  - c. Cell Swizzle: Swizzles the cell positions to other axii. For Example. In XZY mode, the Y and Z coordinates will be swapped, so an input Y coordinate will map to Z instead and vice versa.
3. Changes in the Grid will affect all child Layer GameObjects with the Tilemap, Tilemap Renderer and TilemapCollider2D components.



## Tilemap Palette

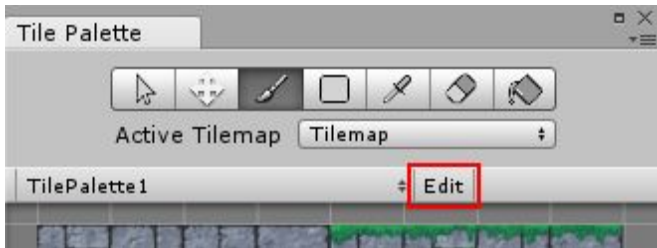


- Alt + Left Mouse Button to pan
- Middle Mouse Button to pan
- Mouse Wheel to zoom in or out
- Left-click to select a tile
- Left-click drag to select multiple tiles

## Editing Tilemap Palette

1. Select the desired palette from the drop-down menu.

2. Click on the Edit button at the side of the palette selection menu.

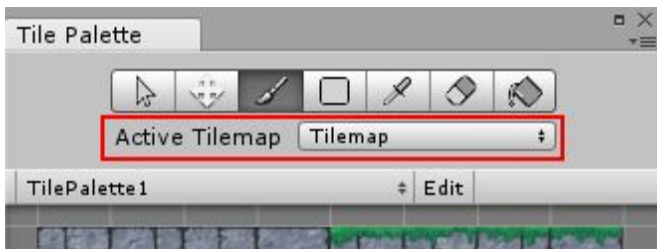


3. Adjust the Palette using the Tile Palette tools just like editing a Tilemap in the scene.
4. When done, click on the edit button to exit editing mode.

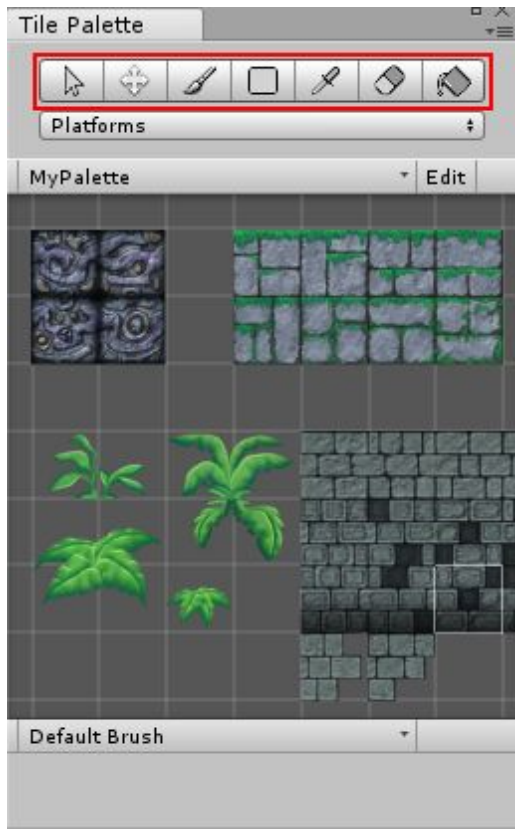
**Remember to save your project to save the Palette!**


## Painting Tilemaps

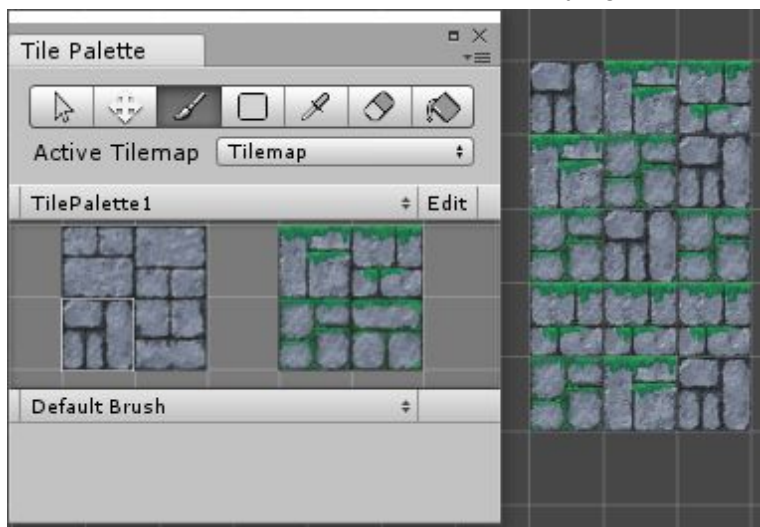
1. To paint on a Tilemap it must be the selected Active Tilemap in the Tile Palette. Tilemaps in the Scene are automatically added to the list.



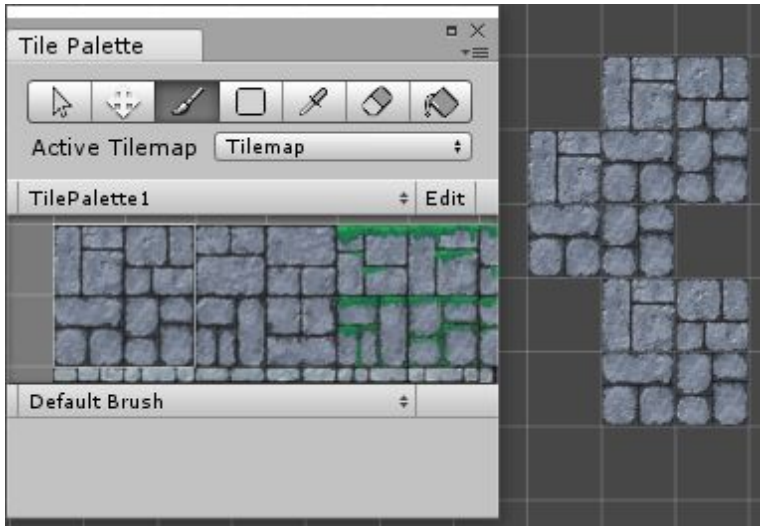
2. The tile painting tools are found on the Tilemap Palette



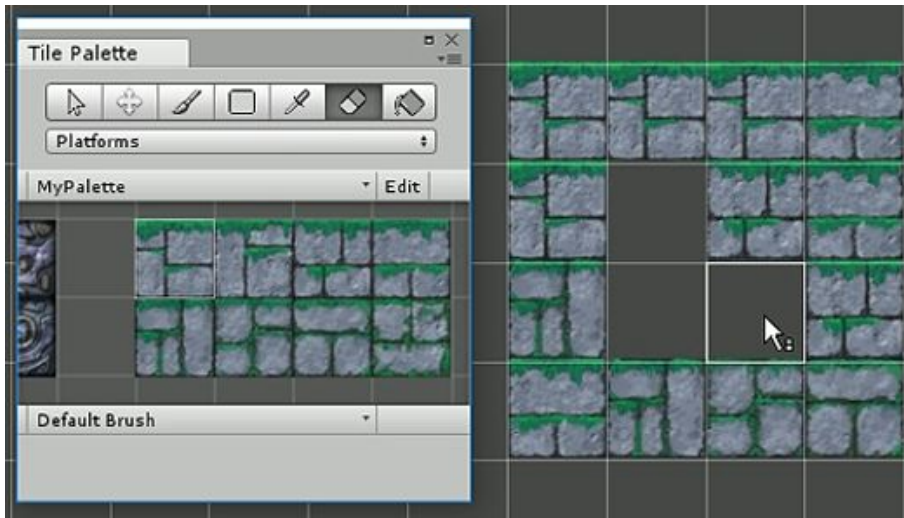
3. Click on the Paint Tool icon , select a tile from the Tilemap Palette and Left-click on the Tilemap in the Scene View to start laying out tiles.




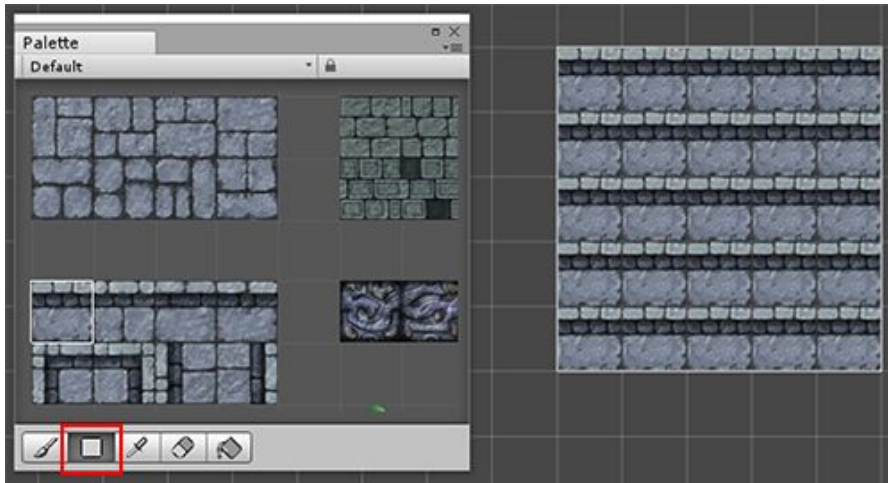
4. A selection of tiles can be painted with the Paint Tool. Left-click and drag in the Tilemap Palette to make a selection.




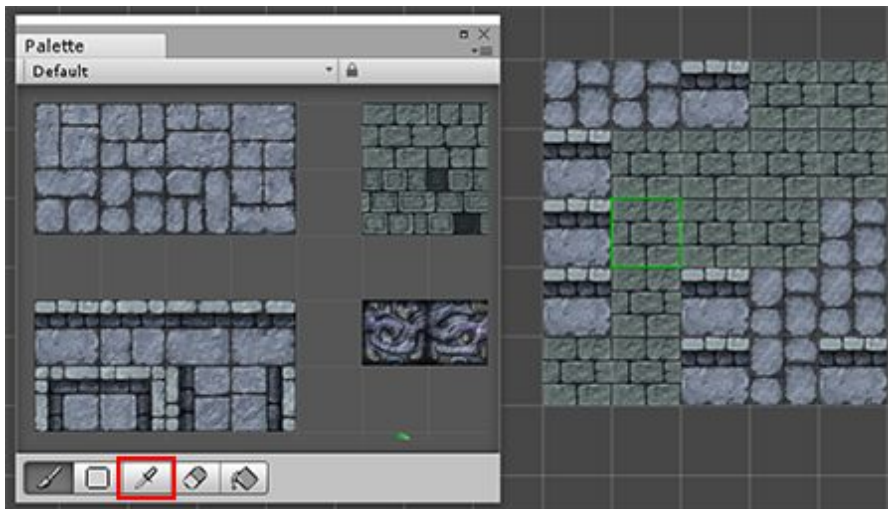
5. Holding Shift while using the Paint Tool will toggle the Erase Tool




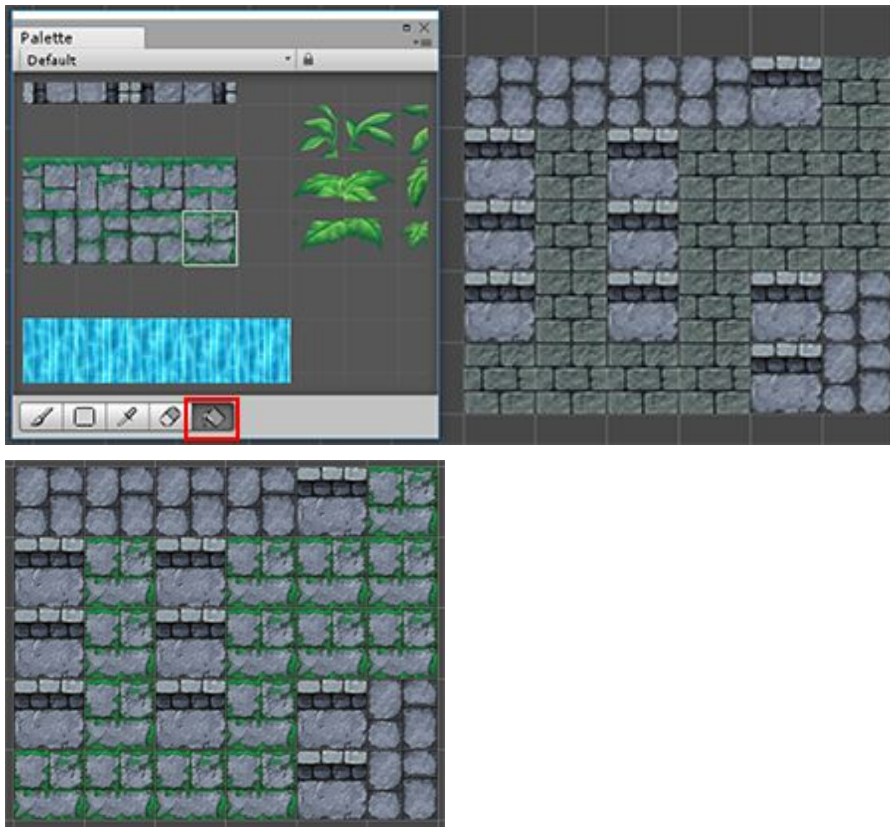
6. The Rectangle tool  draws a rectangular shape on the Tilemap and fills it with the selected tiles.




7. The Picker Tool  is used to pick tiles from the Tilemap to paint with. Left-click and drag to select multiple tiles. Holding the Control key while in Paint Tool mode will toggle the Picker Tool.

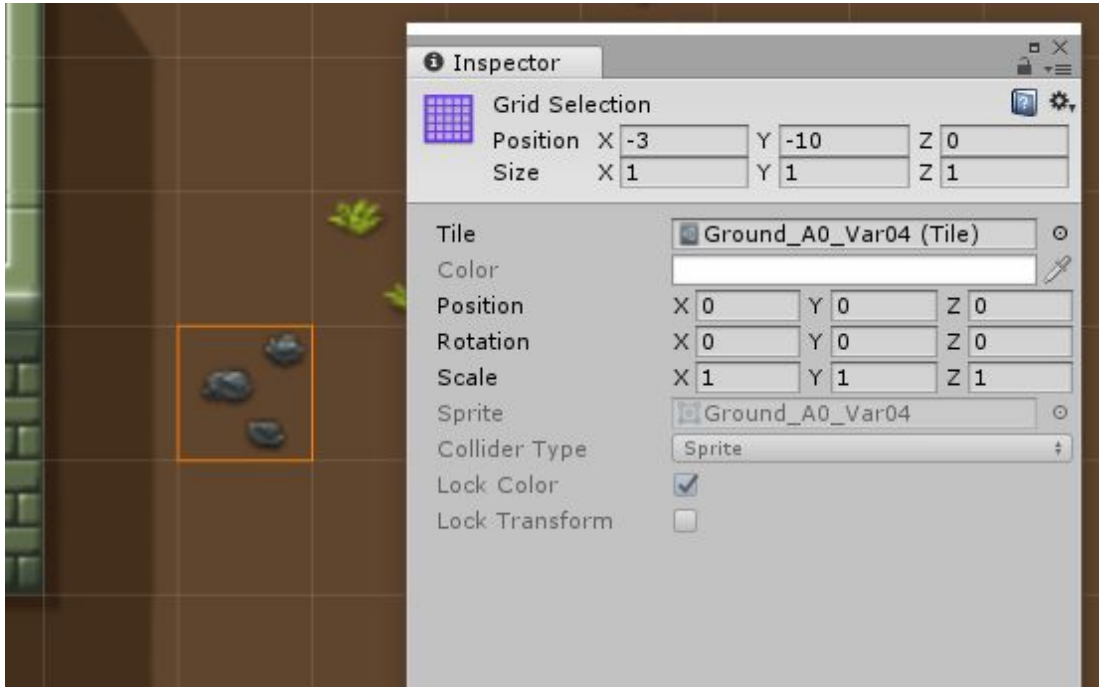



8. The Fill Tool  is used to fill an area of tiles with the selected tile.

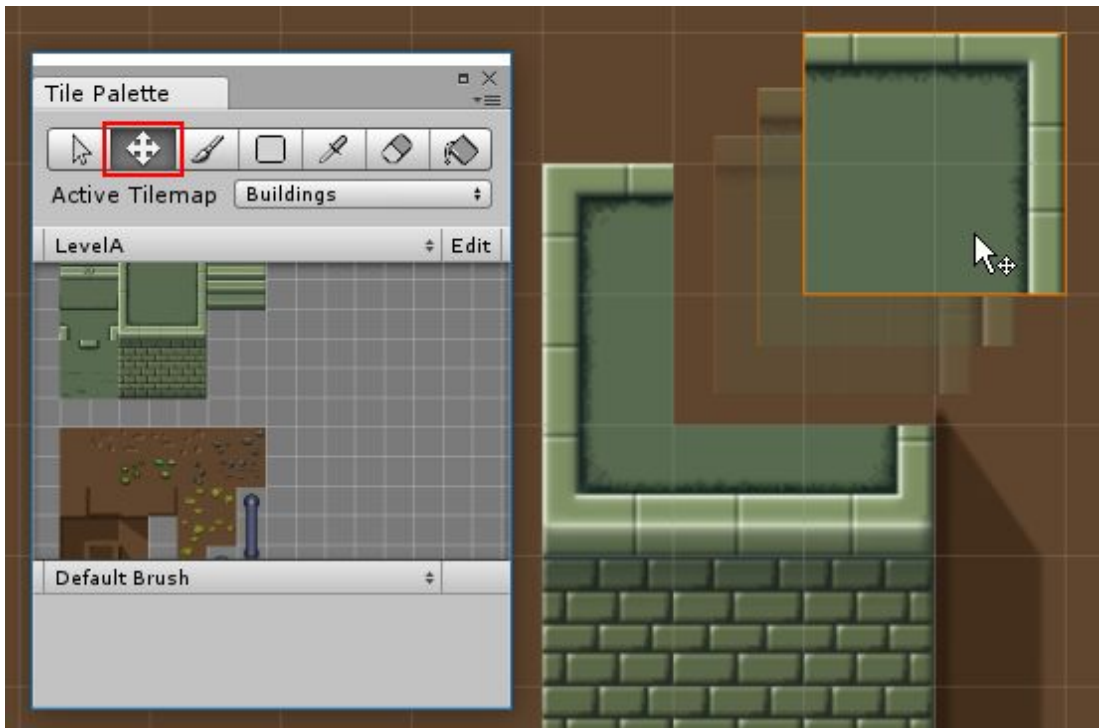




9. The Select Tool  is used to select an area of tiles to be inspected.



10. The Move Tool  is used to move a selected area of tiles to another position. Click and drag the selected area to move the tiles.



## Tilemap Focus mode

If you have many Tilemap layers but would like to work solely on a specific layer to work on, you can focus on that particular layer and block out all other GameObject from view.



1. Select the target Tilemap GameObject from the Active Target dropdown in the Palette window or from the Hierarchy window.
2. In the bottom right corner of the SceneView, there is a Tilemap overlay window.
3. Change the Focus On target in the dropdown:
  - a. None - No GameObject is focused on
  - b. Tilemap - The target Tilemap GameObject will be focused on. All other GameObjects will be faded away. This is good if you want to focus on a single Tilemap layer.
  - c. Grid - The parent Grid GameObject with all its children will be focused on. All other GameObjects will be faded away. This is good if you want to focus on the entire Tilemap as a whole.

## Physics 2D and Tilemaps

- You can add a TilemapCollider2D to the GameObject of a Tilemap to generate a collider based on the Tiles of the Tilemap.



- TilemapCollider2D functions as a Collider2D. You can add Effector2Ds to modify the behavior of the TilemapCollider2D. You can also composite the TilemapCollider2D with a [CompositeCollider2D](#).
- Adding or removing Tiles with ColliderType set to Sprite or Grid will add or remove the Tile's collider shape from the TilemapCollider2D on the next LateUpdate for the TilemapCollider2D. This also happens when the TilemapCollider2D is being composited by a CompositeCollider2D.

## How to create a Scriptable Tile

1. Create a new class inheriting from TileBase (or any useful sub-classes of TileBase like Tile).
2. Override any required methods for your new Tile class. The following are the usual methods you would override:
  - a. RefreshTile determines which tiles in the vicinity will be updated as this Tile is added to the Tilemap.
  - b. GetTileData determines what the Tile will look like on the Tilemap.
3. Create instances of your new class using ScriptableObject.CreateInstance<(Your Tile Class)>(). You may convert this new instance to an asset in the Editor in order to use it repeatedly by calling AssetDatabase.CreateAsset().
4. You can also make a custom editor for your tile. This works the same way as custom editors for scriptable objects.
5. Remember to save your project to ensure that your new Tile assets are saved!

## TileBase

All tiles to be added to the Tilemap must inherit from TileBase. TileBase provides a fixed set of APIs to the Tilemap to communicate its rendering properties. For most cases of the APIs, the location of the tile and the instance of the Tilemap the tile is placed on is passed in as arguments of the API. You may use this to determine any required attributes for setting the tile information.

```
public void RefreshTile(Vector3Int location, ITilemap tilemap)
```

RefreshTile determines which tiles in the vicinity will be updated as this Tile is added to the Tilemap. By default, the TileBase will call `tilemap.RefreshTile(location)` to refresh the tile at the current location. Override this to determine which tiles need to be refreshed due to the placement of the new tile.

Example: There is a straight road and you place RoadTile next to it. The straight road isn't valid anymore. It needs a T-section instead. Unity doesn't automatically know what needs to be refreshed, so RoadTile needs to trigger the refresh onto itself, but also onto the neighboring road.

```
public bool GetTileData(Vector3Int location, ITilemap tilemap, ref TileData tileData)
```

GetTileData determines what the Tile will look like on the Tilemap. See TileData below for more details.

```
public bool GetTileAnimationData(Vector3Int location, ITilemap tilemap, ref TileAnimationData tileAnimationData)
```

GetTileAnimationData determines if the Tile is animated. Return true if there is an animation for the tile, other returns false if there is not.

```
public bool StartUp(Vector3Int location, ITilemap tilemap, GameObject go)
```

StartUp is called for each tile when the Tilemap updates for the first time. You can run any start up logic for tiles on the Tilemap if necessary. The argument go is the instanced version of the object passed in as gameobject when GetTileData was called. You may update go as necessary as well.

## Tile

The Tile class is a simple class that allows a sprite to be rendered on the Tilemap. Tile inherits from TileBase. The following is a description of the methods that are overridden to have the Tile's behaviour.

```
public Sprite sprite;
public Color color = Color.white;
public Matrix4x4 transform = Matrix4x4.identity;
public GameObject gameobject = null;
public TileFlags flags = TileFlags.LockColor;
public ColliderType colliderType = ColliderType.Sprite;
```

These are the default properties of a Tile. If the tile was created by dragging and dropping a Sprite onto the Tilemap Palette, the tile would have the sprite property set as the sprite that was dropped in. You may adjust the properties of the tile instance to get the tile required.

```
public void RefreshTile(Vector3Int location, ITilemap tilemap)
```

This is not overridden from TileBase. By default, it will only refresh the Tile at that location.

```
public override void GetTileData(Vector3Int location, ITilemap tilemap, ref TileData tileData)
```

```
{
    tileData.sprite = this.sprite;
    tileData.color = this.color;
    tileData.transform = this.transform;
    tileData.gameobject = this.gameobject;
    tileData.flags = this.flags;
    tileData.colliderType = this.colliderType;
}
```

This fills in the required information for Tilemap to render the Tile by copying the properties of the Tile instance into tileData.

```
public bool GetTileAnimationData(Vector3Int location, ITilemap tilemap, ref TileAnimationData tileAnimationData)
```

This is not overridden from TileBase. By default, the Tile class does not run any Tile animation and returns false.

```
public bool StartUp(Vector3Int location, ITilemap tilemap, GameObject go)
```

This is not overridden from TileBase. By default, the Tile class does not have any special start up functionality. If `tileData.gameobject` is set, the Tilemap will still instantiate it on start up and place it at the location of the tile.

Other Useful Classes:

### TileFlags

`None = 0`

No flags are set for the tile. This is the default for most tiles.

`LockColor = 1 << 0`

Set this flag if the Tile script controls the color of the Tile. If this is set, the Tile will control the color as it is placed onto the Tilemap. You will not be able to change the tile's color through painting or using scripts. If this is not set, you will be able to change the tile's color through painting or using scripts.

`LockTransform = 1 << 1`

Set this flag if the Tile script controls the transform of the Tile. If this is set, the Tile will control the transform as it is placed onto the Tilemap. You will not be able to rotate or change the tile's transform through painting or using scripts. If this is not set, you will be able to change the tile's transform through painting or using scripts.

`InstantiateSpawnGameObjectRuntimeOnly = 1 << 2`

Set this flag if the Tile script should spawn its game object only when your project is running and not in Editor mode.

`LockAll = LockColor | LockTransform`

This is a combination of all the lock flags used by TileBase.

### Tile.ColliderType

`None = 0`

No collider shape will be generated by this Tile..

`Sprite = 1`

The collider shape generated by this Tile will be the physics shape set by the Sprite that the Tile returns. If no physics shape has been set in the Sprite, it will try to generate a shape based on the outline of the Sprite.

Note: If a collider shape for a Tile needs to be generated in runtime, please do set a physics shape for the Sprite or set the Texture of the Sprite to be readable in order for Unity to generate a shape based on the outline.

`Grid = 2`

The collider shape generated by this Tile will be the shape of the cell defined by the layout of the Grid.

## ITilemap

ITilemap is the base class where Tile can retrieve data from the Tilemap when the Tilemap tries to retrieve data from the Tile.

`Vector3Int origin { get; }`

This returns the origin point of the Tilemap in cellspace.

`Vector3Int size { get; }`

This returns the size of the Tilemap in cellspace.

`Bounds localBounds { get; }`

This returns the bounds of the Tilemap in localspace.

`BoundsInt cellBounds { get; }`

This returns the bounds of the Tilemap in cellspace.

`Sprite GetSprite(Vector3Int location);`

This returns the sprite used by the tile in the Tilemap at the given location.

`Color GetColor(Vector3Int location);`

This returns the color used by the tile in the Tilemap at the given location.

`Matrix4x4 GetTransformMatrix(Vector3Int location);`

This returns the transform matrix used by the tile in the Tilemap at the given location.

`TileFlags GetTileFlags(Vector3Int location);`

This returns the tile flags used by the tile in the Tilemap at the given location.

`TileBase GetTile(Vector3Int location);`

This returns the tile in the Tilemap at the given location. If there is no tile there, it returns null.

`T GetTile<T>(Vector3Int location) where T : TileBase;`

This returns the tile in the Tilemap at the given location with type T. If there is no tile with the matching type there, it returns null.

```
void RefreshTile(Vector3Int location);
```

This requests a refresh of the tile in the Tilemap at the given location.

```
T GetComponent<T>();
```

This returns the component T that is attached to the GameObject of the Tilemap.

## TileData

```
public Sprite sprite
```

This is the sprite that will be rendered for the Tile.

```
public Color color
```

This is the color that will tint the sprite used for the Tile.

```
public Matrix4x4 transform
```

This is the transform matrix used to determine the final location of the Tile. Modify this to add rotations or scaling to the tile.

```
public GameObject gameobject
```

This is the game object that will be instantiated when the Tile is added to the Tilemap.

```
public TileFlags flags
```

These are the flags which controls the Tile's behaviour. See TileFlags above for more details.

```
public Tile.ColliderType colliderType
```

This controls the collider shape generated by the Tile for an attached TilemapCollider2D. See Tile.ColliderType above for more details.

## TileAnimationData

```
public Sprite[] animatedSprites
```

This is an array of sprites for the Tile animation. The Tile will be animated by these sprites in sequential order.

```
public float animationSpeed
```

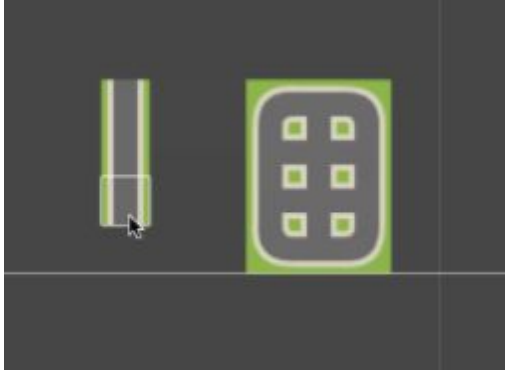
This is the speed where the Tile animation is run. This is combined with the Tilemap's animation speed for the actual speed.

```
public float animationTimeOffset
```

This allows you to start the animation at a different time frame.

## An example Scriptable Tile:

### RoadTile



The RoadTile example will provide the ability to easily layout linear segments onto the Tilemap, such as roads or pipes, with a minimal set of sprites. The following is a script used to create the tile.

```
using UnityEngine;
using System.Collections;

#if UNITY_EDITOR
using UnityEditor;
#endif

public class RoadTile : Tile
{
    public Sprite[] m_Sprites;
    public Sprite m_Preview;

    // This refreshes itself and other RoadTiles that are orthogonally and diagonally
    adjacent
    public override void RefreshTile(Vector3Int location, ITilemap tilemap)
    {
        for (int yd = -1; yd <= 1; yd++)
            for (int xd = -1; xd <= 1; xd++)
            {
                Vector3Int position = new Vector3Int(location.x + xd, location.y
+ yd, location.z);
                if (HasRoadTile(tilemap, position))
                    tilemap.RefreshTile(position);
            }
    }

    // This determines which sprite is used based on the RoadTiles that are adjacent to it
    and rotates it to fit the other tiles.
```

```
// As the rotation is determined by the RoadTile, the TileFlags.OverrideTransform is set for the tile.
```

```
public override void GetTileData(Vector3Int location, ITilemap tilemap, ref TileData tileData)
```

```
{
    int mask = HasRoadTile(tilemap, location + new Vector3Int(0, 1, 0)) ? 1 : 0;
    mask += HasRoadTile(tilemap, location + new Vector3Int(1, 0, 0)) ? 2 : 0;
    mask += HasRoadTile(tilemap, location + new Vector3Int(0, -1, 0)) ? 4 : 0;
    mask += HasRoadTile(tilemap, location + new Vector3Int(-1, 0, 0)) ? 8 : 0;
```

```
    int index = GetIndex((byte)mask);
```

```
    if (index >= 0 && index < m_Sprites.Length)
```

```
    {
        tileData.sprite = m_Sprites[index];
        tileData.color = Color.white;
        tileData.transform.SetTRS(Vector3.zero, GetRotation((byte) mask),
```

```
Vector3.one);
```

```
        tileData.flags = TileFlags.LockTransform;
        tileData.colliderType = ColliderType.None;
```

```
    }
```

```
    else
```

```
    {
```

```
        Debug.LogWarning("Not enough sprites in RoadTile instance");
```

```
    }
```

```
}
```

```
// This determines if the Tile at the position is the same RoadTile.
```

```
private bool HasRoadTile(ITilemap tilemap, Vector3Int position)
```

```
{
```

```
    return tilemap.GetTile(position) == this;
```

```
}
```

```
// The following determines which sprite to use based on the number of adjacent RoadTiles
```

```
private int GetIndex(byte mask)
```

```
{
```

```
    switch (mask)
```

```
    {
```

```
        case 0: return 0;
```

```
        case 3:
```

```
        case 6:
```

```
        case 9:
```

```
        case 12: return 1;
```

```
        case 1:
```

```
        case 2:
```

```
        case 4:
```

```
        case 5:
```

```
        case 10:
```

```
        case 8: return 2;
```

```

        case 7:
        case 11:
        case 13:
        case 14: return 3;
        case 15: return 4;
    }
    return -1;
}

// The following determines which rotation to use based on the positions of adjacent
RoadTiles
private Quaternion GetRotation(byte mask)
{
    switch (mask)
    {
        case 9:
        case 10:
        case 7:
        case 2:
        case 8:
            return Quaternion.Euler(0f, 0f, -90f);
        case 3:
        case 14:
            return Quaternion.Euler(0f, 0f, -180f);
        case 6:
        case 13:
            return Quaternion.Euler(0f, 0f, -270f);
    }
    return Quaternion.Euler(0f, 0f, 0f);
}

#if UNITY_EDITOR
// The following is a helper that adds a menu item to create a RoadTile asset
[MenuItem("Assets/Create/RoadTile")]
public static void CreateRoadTile()
{
    string path = EditorUtility.SaveFilePanelInProject("Save Road Tile", "New Road
Tile", "asset", "Save Road Tile", "Assets");

    if (path == "")
        return;

    AssetDatabase.CreateAsset(ScriptableObject.CreateInstance<RoadTile>(), path);
}
#endif
}

```



## How to create a Scriptable Brush

1. Create a new class inheriting from GridBrushBase (or any useful sub-classes of GridBrushBase like GridBrush).
2. Override any required methods for your new Brush class. The following are the usual methods you would override:
  - a. Paint allows the Brush to add items onto the target Grid
  - b. Erase allows the Brush to remove items from the target Grid
  - c. FloodFill allows the Brush to fill items onto the target Grid
  - d. Rotate rotates the items set in the Brush.
  - e. Flip flips the items set in the Brush.
3. Create instances of your new class using ScriptableObject.CreateInstance<(Your Brush Class)>(). You may convert this new instance to an asset in the Editor in order to use it repeatedly by calling AssetDatabase.CreateAsset().
4. You can also make a custom editor for your brush. This works the same way as custom editors for scriptable objects. The following are the main methods you would want to override when creating a custom editor:
  - a. You can override OnPaintInspectorGUI to have an inspector show up on the Palette when the Brush is selected to provide additional behaviour when painting.
  - b. You can also override OnPaintSceneGUI to add additional behaviour when painting on the SceneView.
  - c. You can also override validTargets to have a custom list of targets which the Brush can interact with. This list of targets will be shown as a dropdown in the Palette window.
5. When created, the Scriptable Brush will be listed in the Brushes Dropdown in the Palette window. By default, an instance of the Scriptable Brush script will be instantiated and stored in the Library folder of your project. Any modifications to the brush properties will be stored in that instance. If you want to have multiple copies of that Brush with different properties, you can instantiate the Brush as assets in your project. These Brush assets will be listed separately in the Brush dropdown.
6. You can add a CustomGridBrush attribute to your Scriptable Brush class. This will allow you to configure the behaviour of the Brush in the Palette window. The CustomGridBrush attribute has the following properties:
  - a. HideAssetInstances - Setting this to true will hide all copies of created Brush assets in the Palette window. Set this if you only want the default instance to show up in the Brush dropdown in the Palette window.
  - b. HideDefaultInstances - Setting this to true will hide the default instance of the Brush in the Palette window. Set this if you only want created assets to show up in the Brush dropdown in the Palette window.
  - c. DefaultBrush - Setting this to true will set the default instance of the Brush as the default Brush in the project. This makes this Brush the default selected Brush

whenever the project starts up. Only set one Scriptable Brush to be the Default Brush!

- d. **DefaultName** - Setting this will make the Brush dropdown use this as the name for the Brush instead of the name of the class of the Brush.
7. Remember to save your project to ensure that your new Brush assets are saved!

## GridBrushBase

All brushes added must inherit from GridBrushBase. GridBrushBase provides a fixed set of APIs for painting.

```
public virtual void Paint(GridLayout grid, GameObject brushTarget, Vector3Int position)
```

Paint adds data onto the target GameObject brushTarget with the GridLayout grid at the given position. This is triggered when the Brush is activated on the grid and the Paint tool is selected on the Palette window. Override this to implement the action desired on painting.

```
public virtual void Erase(GridLayout grid, GameObject brushTarget, Vector3Int position)
```

Erase removes data onto the target GameObject brushTarget with the GridLayout grid at the given position. This is triggered when the Brush is activated on the grid and the Erase tool is selected on the Palette window. Override this to implement the action desired on erasing.

```
public virtual void BoxFill(GridLayout grid, GameObject brushTarget, BoundsInt position)
```

BoxFill adds data onto the target GameObject brushTarget with the GridLayout grid onto the given bounds. This is triggered when the Brush is activated on the grid and the BoxFill tool is selected on the Palette window. Override this to implement the action desired on filling.

```
public virtual void FloodFill(GridLayout grid, GameObject brushTarget, Vector3Int position)
```

FloodFill adds data onto the target GameObject brushTarget with the GridLayout grid starting at the given position and filling all other possible areas linked to the position. This is triggered when the Brush is activated on the grid and the FloodFill tool is selected on the Palette window. Override this to implement the action desired on filling.

```
public virtual void Rotate(RotationDirection direction)
```

Rotate rotates the content in the brush with the given direction based on the currently set pivot.

```
public virtual void Flip(FlipAxis flip)
```

Flip flips the content of the brush with the given axis based on the currently set pivot.

```
public virtual void Select(GridLayout grid, GameObject brushTarget, BoundsInt position)
```

Select marks a boundary on the target GameObject brushTarget with the GridLayout grid from the given bounds. This allows you to view information based on the selected boundary and move the selection with the Move tool. This is triggered when the Brush is activated on the grid and the Select tool is selected on the Palette window. Override this to implement the action desired when selecting from a target.

```
public virtual void Pick(GridLayout grid, GameObject brushTarget, BoundsInt position, Vector3Int pivot)
```

Pick pulls data from the target GameObject brushTarget with the GridLayout grid from the given bounds and pivot position, and fills the brush with that data. This is triggered when the Brush is activated on the grid and the Pick tool is selected on the Palette window. Override this to implement the action desired when picking from a target.

```
public virtual void Move(GridLayout grid, GameObject brushTarget, BoundsInt from, BoundsInt to)
```

Move marks the movement from the target GameObject brushTarget with the GridLayout grid from the given starting position to the given ending position. Override this to implement the action desired when moving from a target. This is triggered when the Brush is activated on the grid and the Move tool is selected on the Palette window and the Move is performed (MouseDown). Generally, this would be any behaviour while a Move operation from the brush is being performed.

```
public virtual void MoveStart(GridLayout grid, GameObject brushTarget, BoundsInt position)
```

MoveStart marks the start of a move from the target GameObject brushTarget with the GridLayout grid from the given bounds. This is triggered when the Brush is activated on the grid and the Move tool is selected on the Palette window and the Move is first triggered (MouseDown). Override this to implement the action desired when starting a move from a target. Generally, this would be picking of data from the target with the given start position.

```
public virtual void MoveEnd(GridLayout grid, GameObject brushTarget, BoundsInt position)
```

MoveEnd marks the end of a move from the target GameObject brushTarget with the GridLayout grid from the given bounds. This is triggered when the Brush is activated on the grid and the Move tool is selected on the Palette window and the Move is completed (MouseUp). Override this to implement the action desired when ending a move from a target. Generally, this would be painting of data to the target with the given end position.

## GridBrushEditorBase

All brush editors added must inherit from GridBrushEditorBase. GridBrushEditorBase provides a fixed set of APIs for drawing of inspectors on the Palette window and drawing of Gizmos on the SceneView.

`public virtual GameObject[] validTargets`

This returns a list of GameObjects which are valid targets to be painted on by the brush. This will show up in the dropdown in the Palette window. Override this to have a custom list of targets which this Brush can interact with.

`public virtual void OnPaintInspectorGUI()`

This displays an inspector for editing Brush options in the Palette. Use this to update brush functionality while editing in the Scene view.

`public virtual void OnSelectionInspectorGUI()`

This displays an inspector for when cells are selected on a target Grid. Override this to show a custom inspector view for the selected cells.

`public virtual void OnPaintSceneGUI(GridLayout grid, GameObject brushTarget, BoundsInt position, GridBrushBase.Tool tool, bool executing)`

This is used for drawing additional gizmos on the Scene View when painting with the brush. Tool is the currently selected tool in the Palette. Executing returns whether the brush is being used at the particular time.

## Other Useful Classes:

### GridBrushBase.Tool

`Select = 0`

Tool for Selection for a GridBrush

`Move = 1`

Tool for Moving for a GridBrush

`Paint = 2`

Tool for Painting for a GridBrush

**Box** = 3

Tool for Boxfill for a GridBrush

**Pick** = 4

Tool for Picking for a GridBrush

**Erase** = 5

Tool for Erasing for a GridBrush

**Floodfill** = 6

Tool for Floodfill for a GridBrush

GridBrushBase.RotationDirection

**Clockwise** = 0

Clockwise rotation direction. Use this when rotating a Brush.

**CounterClockwise** = 1

Counter clockwise rotation direction. Use this when rotating a Brush.

GridBrushBase.FlipAxis

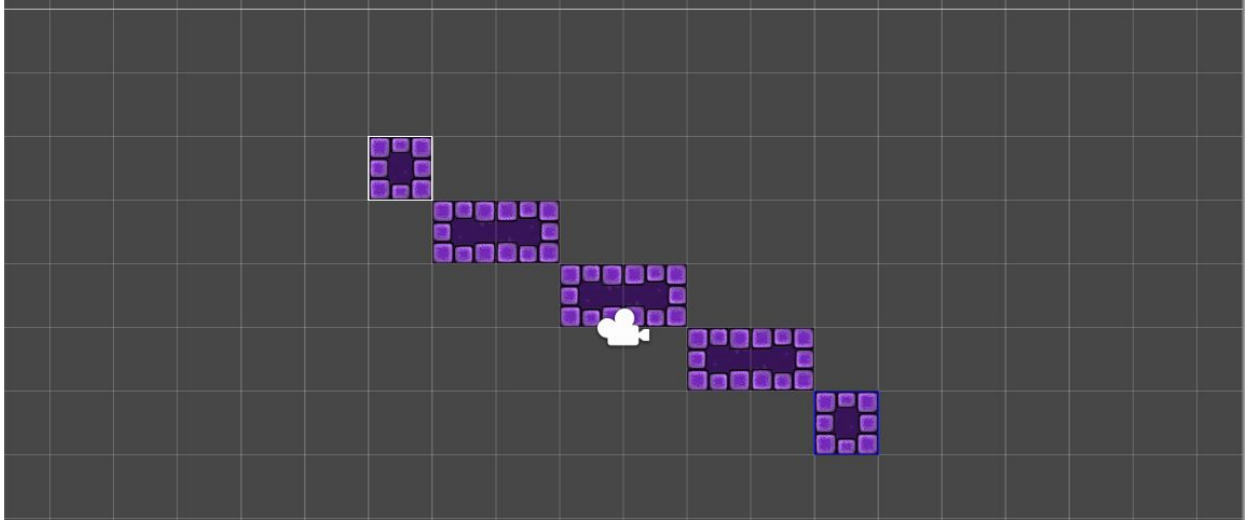
**X** = 0

Flips along the X Axis. Use this when flipping a Brush.

**Y** = 1

Flips along the Y Axis. Use this when flipping a Brush.

A Scriptable Brush example:



LineBrush will provide the ability to easily draw lines of Tiles onto the Tilemap by specifying the start point and the end point. The Paint method for the LineBrush is overridden to allow the user to specify the start of a line with the first mouse click in Paint mode and draw the line with the second mouse click in Paint mode. The OnPaintSceneGUI method is overridden to produce the preview of the line to be drawn between the first and second mouse clicks. The following is a script used to create the Brush.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;

namespace UnityEditor
{
    [CustomGridBrush(true, false, false, "Line Brush")]
    public class LineBrush : GridBrush {
        public bool lineStartActive = false;
        public Vector3Int lineStart = Vector3Int.zero;

        public override void Paint(GridLayout grid, GameObject brushTarget, Vector3Int
position)
        {
            if (lineStartActive)
            {
                Vector2Int startPos = new Vector2Int(lineStart.x, lineStart.y);
                Vector2Int endPos = new Vector2Int(position.x, position.y);
                if (startPos == endPos)
                    base.Paint(grid, brushTarget, position);
                else
                {
                    foreach (var point in GetPointsOnLine(startPos, endPos))
```

```

        {
            Vector3Int paintPos = new Vector3Int(point.x, point.y, position.z);
            base.Paint(grid, brushTarget, paintPos);
        }
    }
    lineStartActive = false;
}
else
{
    lineStart = position;
    lineStartActive = true;
}
}

[MenuItem("Assets/Create/Line Brush")]
public static void CreateBrush()
{
    string path = EditorUtility.SaveFilePanelInProject("Save Line Brush", "New Line
Brush", "asset", "Save Line Brush", "Assets");

    if (path == "")
        return;

    AssetDatabase.CreateAsset(ScriptableObject.CreateInstance<LineBrush>(), path);
}

// http://ericw.ca/notes/bresenhams-line-algorithm-in-csharp.html
public static IEnumerable<Vector2Int> GetPointsOnLine(Vector2Int p1, Vector2Int p2)
{
    int x0 = p1.x;
    int y0 = p1.y;
    int x1 = p2.x;
    int y1 = p2.y;

    bool steep = Math.Abs(y1 - y0) > Math.Abs(x1 - x0);
    if (steep)
    {
        int t;
        t = x0; // swap x0 and y0
        x0 = y0;
        y0 = t;
        t = x1; // swap x1 and y1
        x1 = y1;
        y1 = t;
    }
    if (x0 > x1)
    {
        int t;
        t = x0; // swap x0 and x1
        x0 = x1;

```

```

        x1 = t;
        t = y0; // swap y0 and y1
        y0 = y1;
        y1 = t;
    }
    int dx = x1 - x0;
    int dy = Math.Abs(y1 - y0);
    int error = dx / 2;
    int ystep = (y0 < y1) ? 1 : -1;
    int y = y0;
    for (int x = x0; x <= x1; x++)
    {
        yield return new Vector2Int((steep ? y : x), (steep ? x : y));
        error = error - dy;
        if (error < 0)
        {
            y += ystep;
            error += dx;
        }
    }
    yield break;
}
}

[CustomEditor(typeof(LineBrush))]
public class LineBrushEditor : GridBrushEditor
{
    private LineBrush lineBrush { get { return target as LineBrush; } }

    public override void OnPaintSceneGUI(GridLayout grid, GameObject brushTarget,
BoundsInt position, GridBrushBase.Tool tool, bool executing)
    {
        base.OnPaintSceneGUI(grid, brushTarget, position, tool, executing);
        if (lineBrush.lineStartActive)
        {
            Tilemap tilemap = brushTarget.GetComponent<Tilemap>();
            if (tilemap != null)
                tilemap.ClearAllEditorPreviewTiles();

            // Draw preview tiles for tilemap
            Vector2Int startPos = new Vector2Int(lineBrush.lineStart.x,
lineBrush.lineStart.y);
            Vector2Int endPos = new Vector2Int(position.x, position.y);
            if (startPos == endPos)
                PaintPreview(grid, brushTarget, position.min);
            else
            {
                foreach (var point in LineBrush.GetPointsOnLine(startPos, endPos))
                {
                    Vector3Int paintPos = new Vector3Int(point.x, point.y, position.z);

```



```

        PaintPreview(grid, brushTarget, paintPos);
    }
}

if (Event.current.type == EventType.Repaint)
{
    var min = lineBrush.lineStart;
    var max = lineBrush.lineStart + position.size;

    // Draws a box on the picked starting position
    GL.PushMatrix();
    GL.MultMatrix(GUI.matrix);
    GL.Begin(GL.LINES);
    Handles.color = Color.blue;
    Handles.DrawLine(new Vector3(min.x, min.y, min.z), new Vector3(max.x,
min.y, min.z));
    Handles.DrawLine(new Vector3(max.x, min.y, min.z), new Vector3(max.x,
max.y, min.z));
    Handles.DrawLine(new Vector3(max.x, max.y, min.z), new Vector3(min.x,
max.y, min.z));
    Handles.DrawLine(new Vector3(min.x, max.y, min.z), new Vector3(min.x,
min.y, min.z));
    GL.End();
    GL.PopMatrix();
}
}
}
}
}

```

## 2D Extras in GitHub

Download more example Brushes and Tiles from the 2D Extras GitHub repository.

<https://github.com/Unity-Technologies/2d-extras>



