== UniRPG Ed ==
By Leslie Young
www.plyoung.com

Visit the Unity3D Forum for tutorial videos and support.

Table of Contents

Introduction

UniRPG Ed. is a tile-based editor extension for Unity3D. It allows you to setup tile sets that can then be used to quickly draw maps, like dungeons and terrain, and place items on these maps.

NOTE: The folder for UniRPG Tile Ed. Was changed from "/Assets/UniRPG" to "/Assets/UniRPG TileEd" during the 1.7 update. Importing the new package will however not make this update and you need to do it manually.

UNITY 4 USERS PLEASE NOTE

DLLs compatible with Unity 4 are provided in the file,
\Assets\UniRPG\Documentation\for_unity4.zip

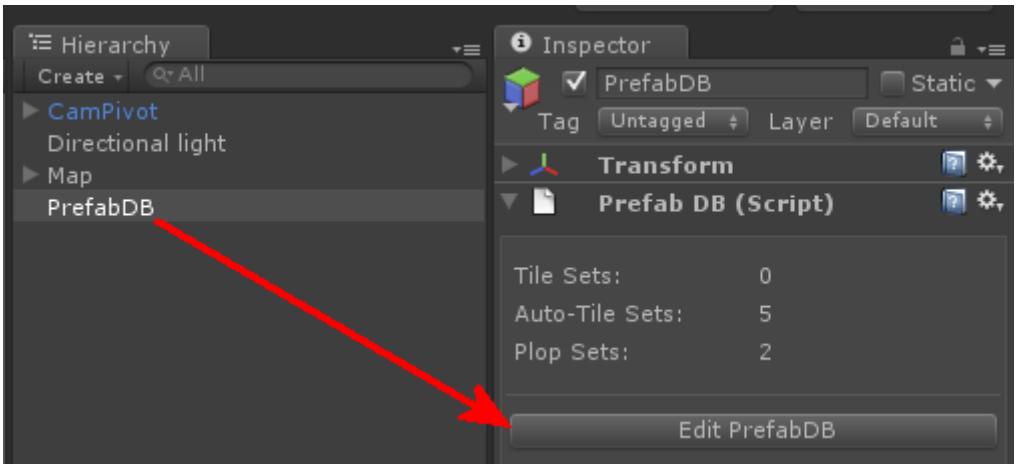
Extract the files to your /Assets/ folder, overwriting the current DLL and MDB files,
\Assets\UniRPG TileEd\Scripts\UniRPGRuntime.dll
\Assets\UniRPG TileEd\Scripts\UniRPGRuntime.dll.mdb
\Assets\UniRPG TileEd\Editor\Scripts\UniRPGEditor.dll
\Assets\UniRPG TileEd\Editor\Scripts\UniRPGEditor.dll.mdb

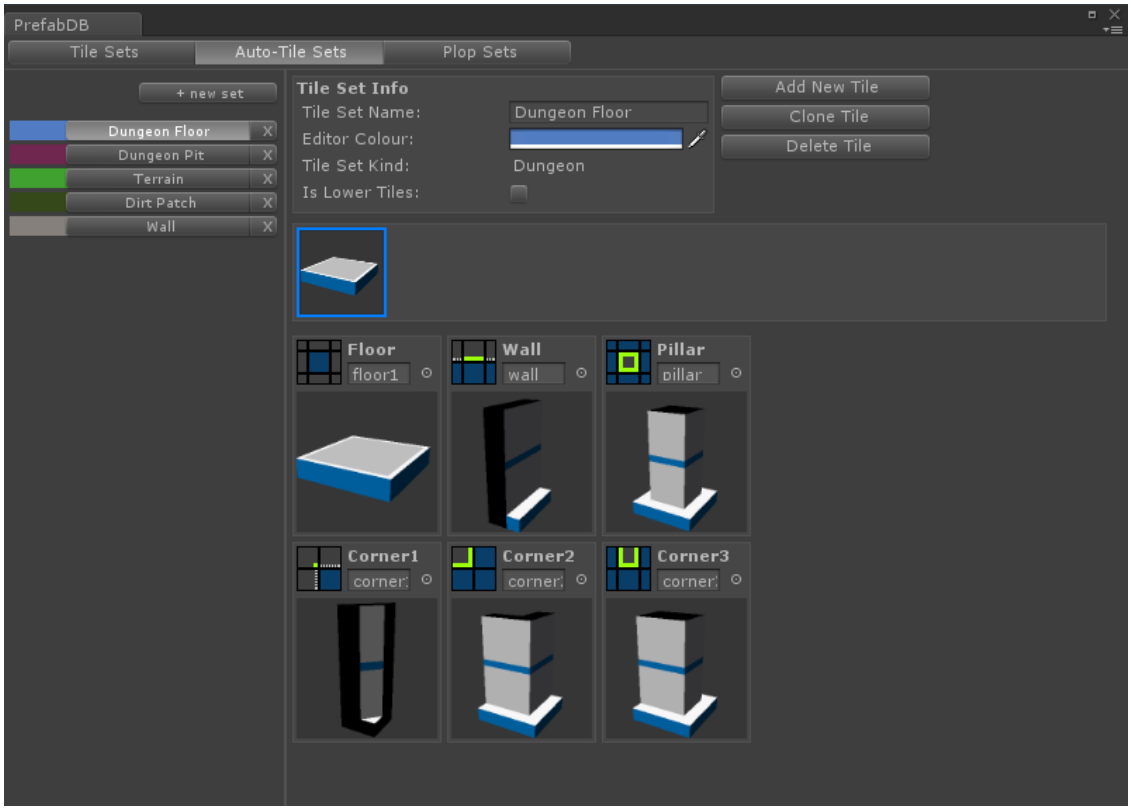
The PrefabDB

The PrefabDB is a container for all tiles and objects to be used on the maps. You need to create a PrefabDB before you can start drawing maps.

From the menu, select GameObject → Create → UniRPG → Create PrefabDB.
A new PrefabDB object will be created in the scene. If you want, you may save this as a prefab and edit the prefab rather than the object in the scene.

When you created the PrefabDB a new editor window appeared where you can edit the PrefabDB. You can also click on a PrefabDB object to find a button in the inspector that will open this editor window.





Tiles and Plops

The PrefabDB is divided into 3 section: Tiles, Auto-Tiles and Plops.

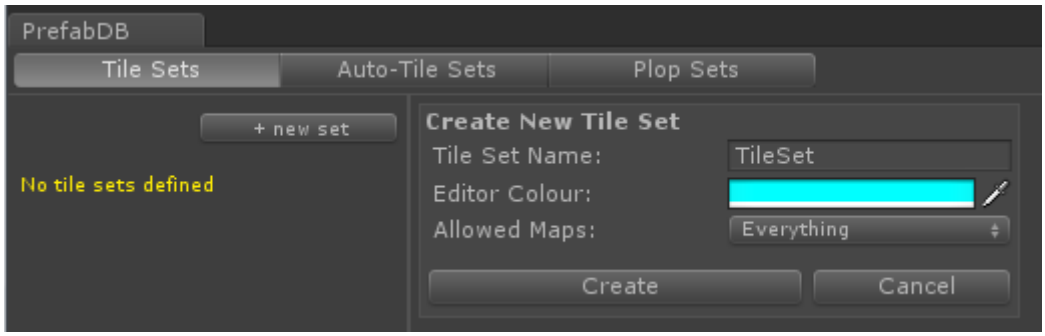
The **Tiles** section allows you define tile sets that can be used to draw on the maps. These can be anything, like floors, walls, doors, etc. and is normally used where Auto-Tiles can't be used.

Auto-Tiles are tiles that knows how to draw their own walls, corners, and cliff-faces as you place new floor tiles on a map and require the various tile-pieces to function correctly.

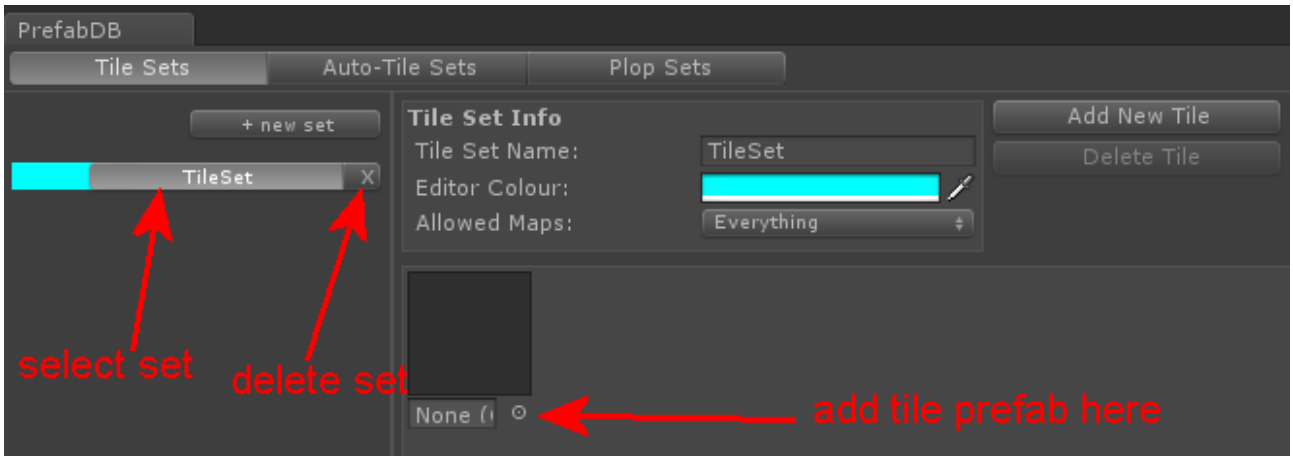
Plops are anything else that can be “plopped” (placed) on a map, like monsters, items, traps, etc.

Defining a new Tile Set

Select the “Tile Sets” tab in the PrefabDB editor window. Click on “new set” to create a new set. You can enter a name for this set, choose a colour that that is used in the editor and choose on what kind of maps this set can be used. There are two kinds of maps (version 1.0), Dungeons and Terrain. You can choose that this tile set can be used on only one or more of these map kinds.



Click on “Create” when you are ready. You will notice that sets are listed on the left-hand panel of the PrefabDB editor window. Click on a set's button to make it active when you have more than one set. The little “x” button next to the name of the set is used to delete the whole set.

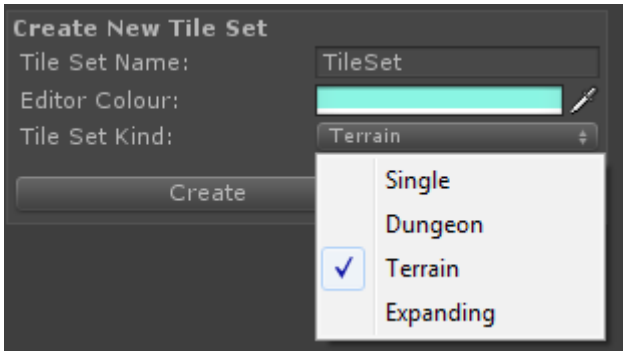


Now that you have a new set, you may start adding tiles by clicking on the “Add New Tile” button. This will create a new empty tile in the set. You can drag a tile prefab from the project panel to the empty tile space or click on the little button below the empty square to select the tile prefab.

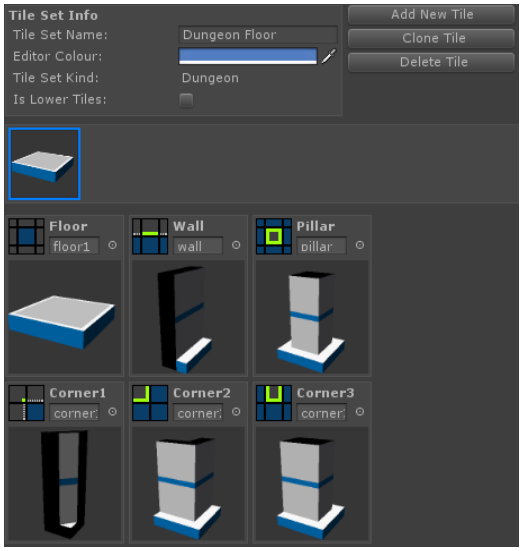
Defining a new Auto-Tile Set

Auto-tiles are a bit more involved than normal tiles since they require various pieces that will be used when drawing the floors of your dungeon or terrain.

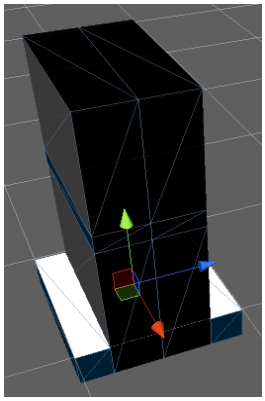
When creating an Auto-Tile set you need to decide what kind of Auto-Tiles kind you want. There will be a drop-down from which you can select either, Dungeon, Terrain and Expanding. Single should not be used in this case as it only apply to normal tiles.



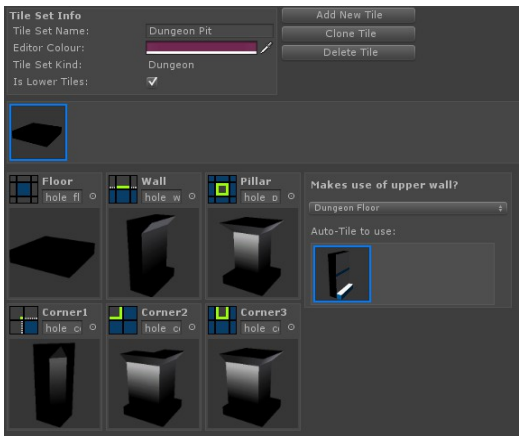
The set kind you choose will determine what tile pieces are required by tiles of the set and how they will act when you draw floors on a map.



Dungeon is used where you want to create Dungeons with floor, walls and corners. Dungeons also allow for floor one level lower than the normal floor, which is typically used for pits/holes, water, lava, etc. tiles. The option “is lower tiles” is used to specify that the set is used for lower floor tiles of a Dungeon.

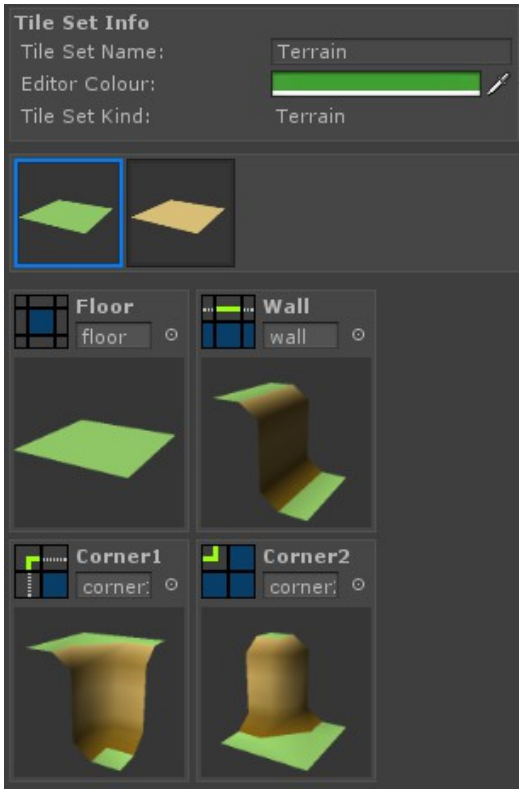


6 tile pieces are required, a floor that fills a whole tile space, and walls and corners that will share tile space with each other. The wall for example should be such a size that two walls could be placed back to back on the same tile.



The little icons (blue/green) should give you an idea of what kind of tile-pieces shape is expected but please have a look at the provided sample models to see how to correctly model these pieces and how they all fit together.

“Lower type” Dungeon tiles has an additional option to specify which tile-set to use for the “upper” walls of the lower tiles when the lower tiles are not surrounded by normal/upper floors.

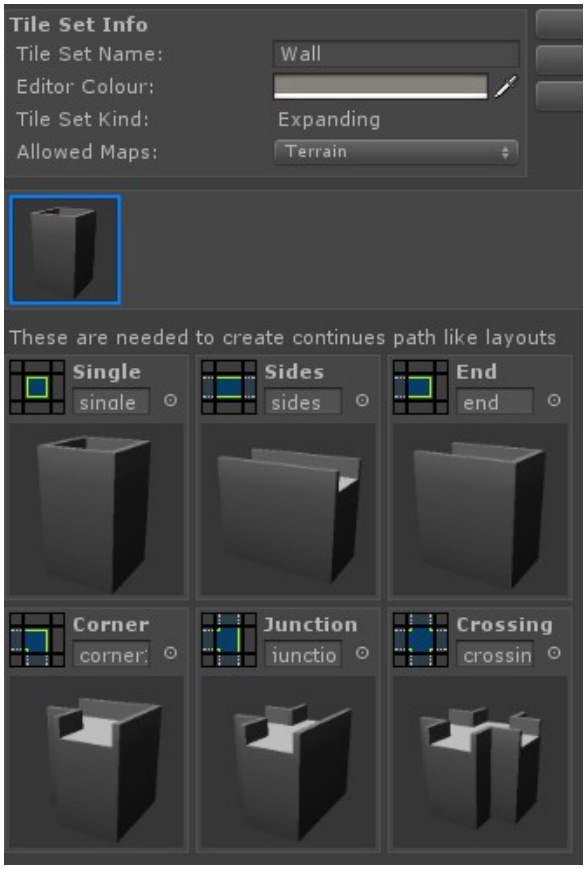


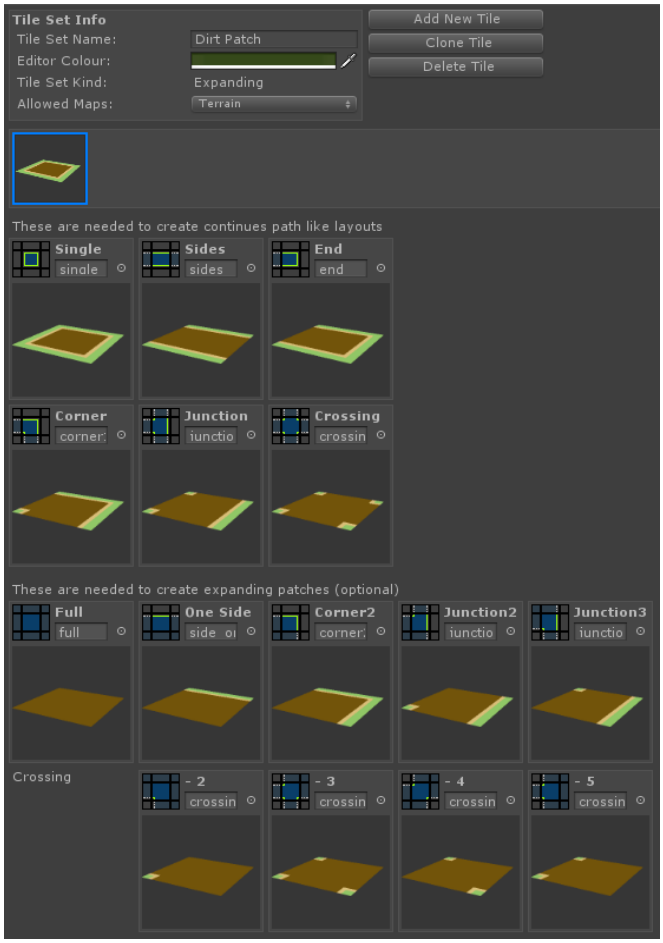
Terrain is used to draw flat terrain floors and in the case of raised or lowered terrain, cliffs. Terrain allows for placement of tiles up to 10 levels higher or lower from the grid origin.

Terrain requires only 4 pieces to function but each piece will take up a whole tile space. Have a look at the provided sample models to get an idea of how these pieces fits together at the floor, wall and corners.

Expanding tiles are used for things like paths, walls and patches of dirt, grass, etc. The 1st six pieces of Expanding tiles are required to create paths and walls with corners and intersections while the 9 remaining pieces are optional and used when you want to create patches; that is, areas that are more than one tile wide.

Like with normal tiles you may specify on which kind of maps these tiles can be used.

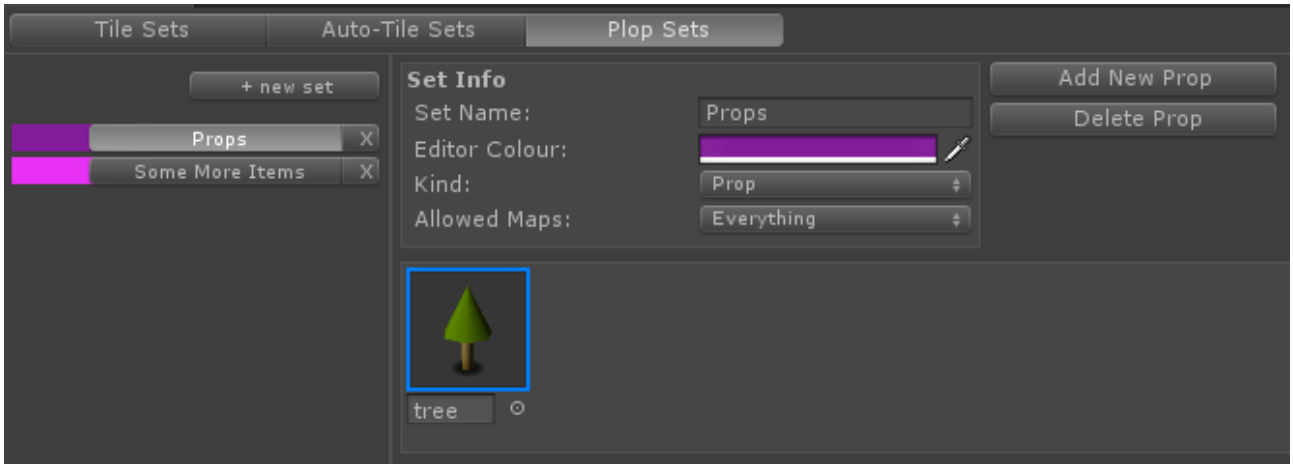




Defining a new Plop Set

Plops are non-tile objects, like monsters, potions, player characters, trees, etc. When creating a new Plop set you will have the option to choose in which kind of maps the plops can be used and what kind of plops you are defining.

The Plop Kind does not have any function under UniRPG Ed. (version 1.0) and is only used to group plops. They might have a function in later versions of UniRPG Ed. and/or other packages that require UniRPG Ed. to function.

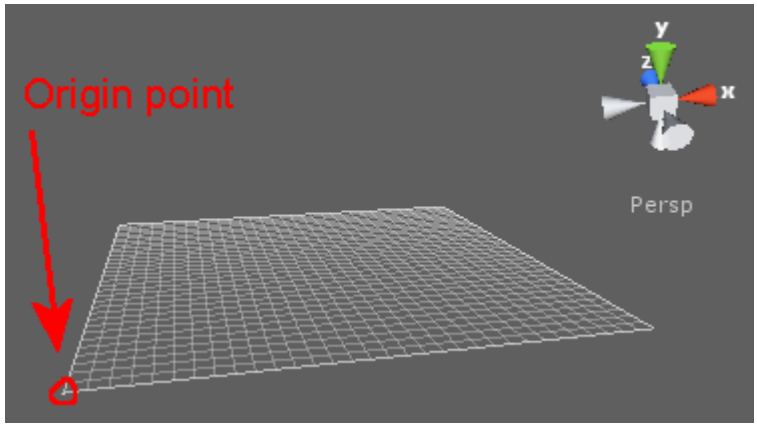


Create a Map

Now that you have a PrefabDB you can start painting maps. From the menu choose, GameObject → Create → UniRPG → Create New Map. A window will pop-up with some options.

Here you can give your new map a **name**, which is useful if you are going to have more than one map in the same scene.

You need to choose a **kind** of map. The options are Dungeon and Terrain and they are painted differently as explained with the PrefabDB setup notes.



The **Map Size** indicates how many tiles you want in the map grid. **Tile Size** is the size of the tiles (floor tiles) in Unity3D units (meters) and the **Tile Height** will determine how low the lower floor of a Dungeon will be or how height the grid of a Terrain map can be moved up and down (when creating cliffs).

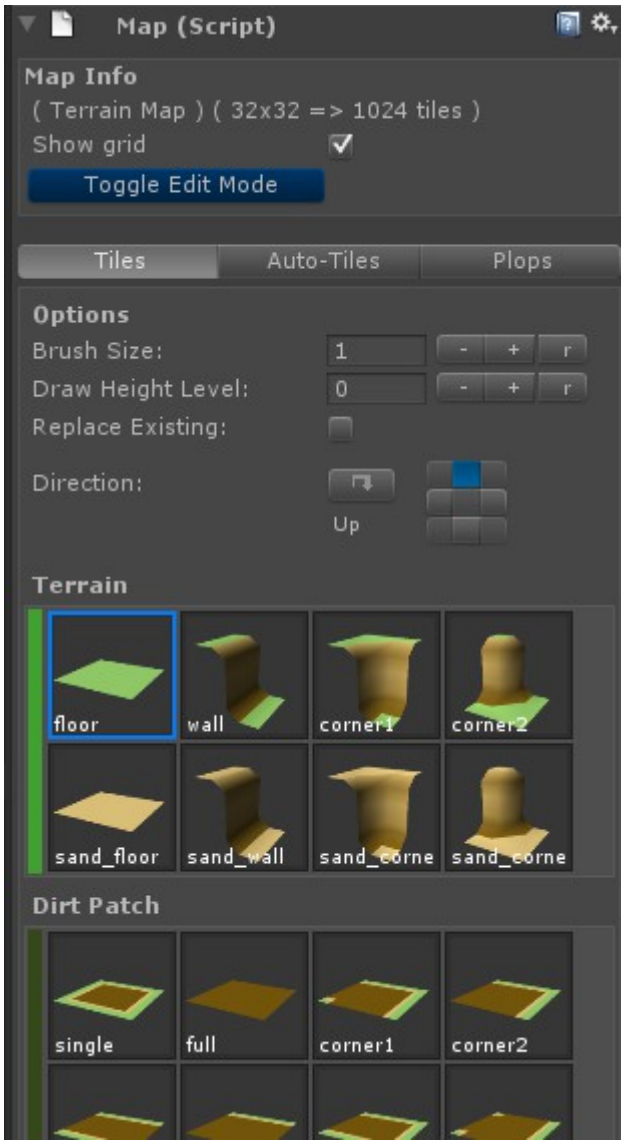
The last option is to specify the PrefabDB containing all the tiles and plops that can be used on the map. This can either be an object in the scene or a prefab.

Hit “Create” to create the new map object in the scene and you should see a grid representing your new map (in scene view). The origin point, the first tile, is indicated by a small white gizmo in one of the corners of the grid. The map expands to the right and upwards from this.

Map Inspector/ Painting Tiles

With the map GameObject selected you will see various options, in the Inspector panel, which will allow you to paint tiles on the grid and place plops.
You will notice the same three tabs that was available in the PrefabDB editor window. These allow you to choose which tool you would like to work with, the Normal Tiles, Auto-Tiles or Plops.

Normal Tiles



Brush size is how many tiles (on the grid) you would like to paint at the same time. This can be a size of up to 5x5.

Draw Height Level is how heigh or low the grid should be from the origin and will determine at what height the tiles will be painted. (terrain maps only)

With **Replace Existing** active, existing tiles will be replaced by whatever new tiles you paint.

Direction is used to rotate the tile at 90º

A list of Normal-Tiles and Auto-Tiles will follow. Note that you will only see the tiles that may be used on the specified map kind (Dungeon or Terrain), as specified in the PrefabDB editor window.

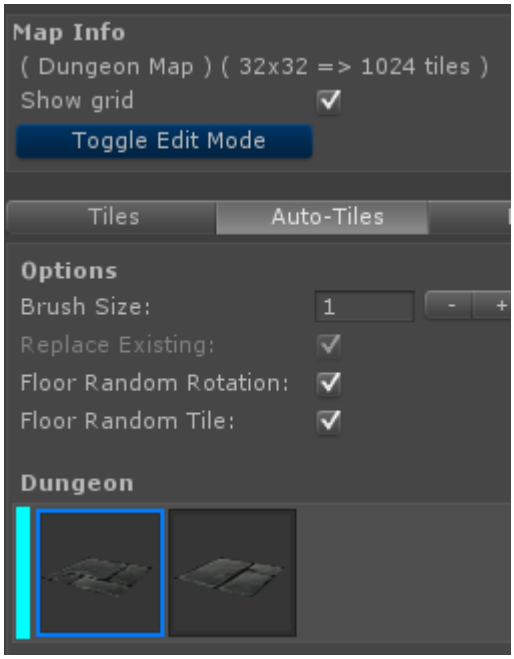
Shortcut Keys

Ctrl+Up/Down arrow keys to move grid up/down (terrain only)

Ctrl+Left/Right arrow keys to rotate the active tile.

Left-mouse button to paint tiles and Right-mouse button to delete tiles.

Auto-Tiles



Brush size is how many tiles (on the grid) you would like to paint at the same time. This can be a size of up to 5x5.

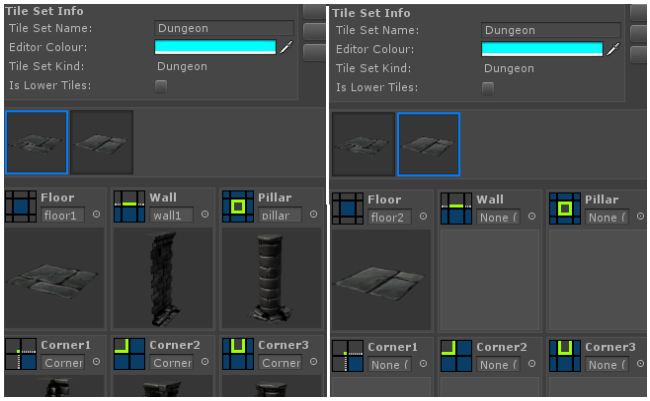
Draw Height Level is how heigh or low the grid should be from the origin and will determine at what height the tiles will be painted. (terrain maps only)

Raise/Lower Mode is used to quickly create higher or lower terrain tiles. With this mode active, when painting on existing tiles will create a tile one level higher than where the grid is and painting with the right-mouse button will create a tile one level lower. (terrain maps only)

The **Replace Existing** option can't be changed since auto-tiles will always modify the tiles on the spot where you draw and around it.

Floor Random Rotation will randomly rotate floor pieces by 90º while you are painting. This is useful to get variation.

Floor Random Tile will randomly select a floor piece from tiles in the same set. You can have any of the tiles in the set selected. You do not have to define wall/corner pieces for the tiles that are only there for the randomiser.



A list of Auto-Tiles will follow. Note that you will only see the tiles that may be used on the specified map kind (Dungeon or Terrain), as specified in the PrefabDB editor window.

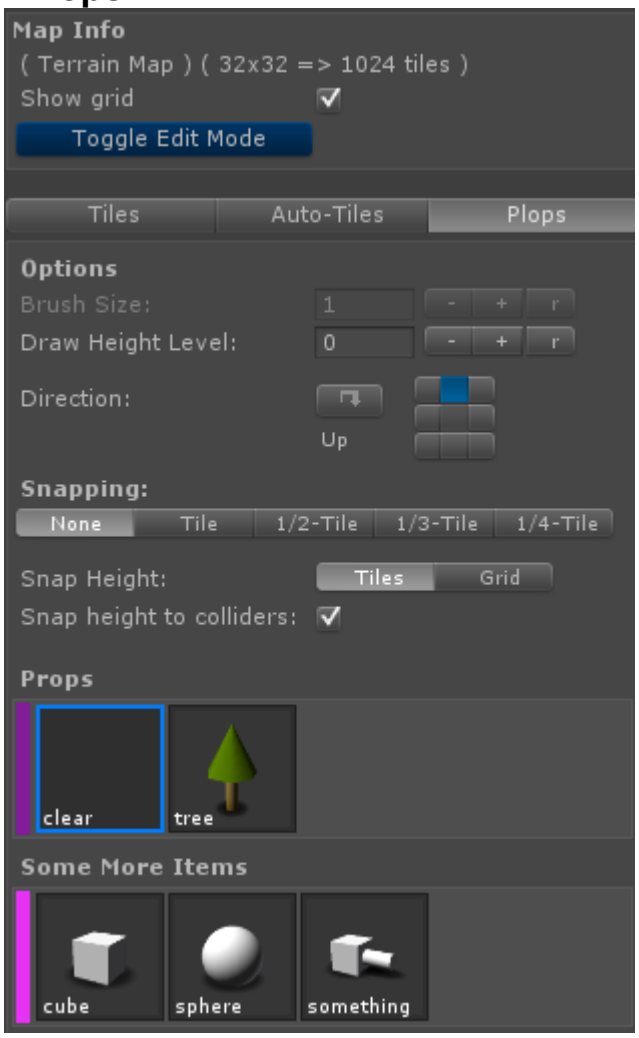
Shortcut Keys

Ctrl+Up/Down arrow keys to move grid up/down (terrain only)

Tab to enable/disable Raise/Lower mode.

Left-mouse button to paint tiles and Right-mouse button to delete tiles (in Raise/Lower mode the right-mouse button will lower existing tiles rather than delete them).

Plops



Brush Size is not available for Plops.

Draw Height Level is how heigh or low the grid should be from the origin and will determine at what height the tiles will be painted. (terrain maps only)

Direction is used to rotate the plop by 45º

Snapping options
You can choose to have no grid/tile snapping or to snap at full, ½, ¼, etc. tile sizes spaces.

Snap Height will allow you to place a plop on Tile or Grid height.

Snap height to colliders is used when you want plops to adjust their height to existing plops in the same spot. Note that this will only work if the plops has colliders.

The various plops defined in the PrefabDB editor window follow with only those allowed on the current map kind, Dungeon or Terrain, shown.

Shortcut Keys

Ctrl+Up/Down arrow keys to move grid up/down (terrain only)

Ctrl+Left/Right arrow keys to rotate the active plop.

Left-mouse button to place plops.

Creating new Tiles/Plops

See the document, UniRPG_Making_Tiles.pdf for help on this subject.

API

There are a few classes and functions you may use at runtime to paint tiles. This could be used to create your ow in-game editor or even in your random dungeon/world generator.

All source code is provided, so you can have a look at the classes to get an idea of how functions are used but here's a run down of what is available at runtime.

Map.cs

The main script that holds info about the map. It has all the functions needed to create maps and place tiles pieces.

Map.tiles is a variable you can use to find all tiles placed in the map. Null if there is no tile present at the queried spot. It is one dimensional array, so to find the correct index into it, if you have an X and Y position, you use the formula, index = y * width_of_map + x;

This is an array of GameObjects, each containing the TilePiece components, so use GetComponent<TilePiece>() to get the actual component if there is a tile an work with that.

Each TilePiece will have various variables to describe it, like its facing direction, if it is a wall, corner, or floor, and a list of all the tile pieces that might be sharing a tile space with it.

One tile can consist of more than one piece, that is why the base tile is of TilePiece type. Think for example about a dungeon map where one tile spot could contain anything from one wall piece to 4 other pieces, like a combination of wall and corner pieces, all occupying the same tile spot to complete the desired effect. Even a floor piece could share a tile with walls and corners, for example the lower walls (under the floor) used as walls for a lower floor tile next to the normal height floor. Paint a few Dungeon maps to get an idea of what this means.

public static Map **CreateNewMap**(string name, UniRPG.MapKind kind, int w, int l, float tileSize, float tileHeight, PrefabDB db)

Used to create a new Map GameObject.
name: name of the object
kind: Terrain or Dungeon map
w & l: width and length of the map in tiles (default is 32x32)
tileSize: size of a tile, default is 1f
tileHeight: height of a tile, default is 1f
db: the prefabDB that contains all the tile pieces

public void **RemoveAllTiles**()

Will remove all the tiles from the map.

private void **DestroyTile**(int idx)

Removes the tile at the specified index. idx = y * width_of_map + x;

private void **DestroyLinkedPiecesIn**(TilePiece mainPiece)

Removes the tile pieces linked to the given piece, but not the main piece itself.

private void **DestroyLinkedPiecesOfFromSetIn**(TilePiece mainPiece, int setIdx)

Same as DestroyLinkedPiecesIn but will only remove pieces that comes from the specified set.

public void **RemoveTile**(int x, int y, int brushSize=1)

Removes the tile at the specified location. Brush size indicates a radius around the specified location that should be included. Do not use brushes bigger that 5 as it can become slow.

public void **FloodFillTile**(int floorLevel, int setIdx, int tileIdx, TilePiece.Direction faceDirection = TilePiece.Direction.Up)

Fills the map with the specified tile. This is a slow process. All existing tiles will be removed. This floodfill from the Normal Tiles. There is a different function for flooding with tiles from Auto-tiles sets.

floorLevel: at what height to fill, 0 is default floor height. Only applies to terrain maps.
setIdx: the set to use
tileIdx: the tile to use from the specified set

public void **PlaceTile**(int x, int y, int floorLevel, int setIdx, int tileIdx, bool replaceExisting = true, TilePiece.Direction faceDirection = TilePiece.Direction.Up, int brushSize = 1, bool fromAutoTiles = false, TilePiece.Kind pieceType = TilePiece.Kind.None)

Used to place a tile piece.

x & y: is the location to place it.
floorLevel: The height index to place tile at, does not apply for Dungeon map, only Terrain map.
setIdx: set to use
tileIdx: tindex of tile to use from set
replaceExisting: repalce existing tile pieces?
faceDirection: direction to face the tile, normally used with corners and walls
brushSize: you can place 1 or more tiles at the same time around the specified x&y. Don;t use brushes of moe than size 5 as it will become slow.
fromAutoTiles: use the “normal” or auto tiles? There is a difference between the two as described in the documentation. This fucntion is used to palce normal tiles but allows you to place a tile from the auto-tile sets as if it was a “normal” tile.. a tile that does not update its surrounding tiles.
pieceType: this is used by the auto tile placement functions.

public void **RemoveAutoTile**(int x, int y, int brushSize=1, int selectedSet=-1)

This removes an auto tile, meaning it will also update the walls around the removed tile.

public void **FloodFillAutoTile**(int floorLevel, int setIdx, int tileIdx, TilePiece.Kind withType = TilePiece.Kind.Floor, TilePiece.Direction faceDirection = TilePiece.Direction.Up)

Similar to **FloodFillTile**, but fills the map with tiles from the auto-tile sets

public void **PlaceAutoTile**(int x, int y, int floorLevel, int setIdx, int tileIdx, int brushSize = 1, bool replaceCurrTile = false, bool floorRandomRot = false, bool floorRandomTile = false)

Similar to **PlaceTile**, but used to place tiles from the auto-tile sets. Auto-tiles also knows how to draw their own walls and corners. So you will specify a floor and the walls around it will be drawn too.

public GameObject **PlacePlop**(Vector3 position, TilePiece.Direction direction, int setIdx, int plopIdx)

Used to place a plop at the specified location.
direction: that it should point in.
setIdx: index of plop set to use
plopIdx: index of the plop in the set

PrefabDB

This is the container for all tiles and plops that can be used on maps. This is setup during design time in the Unity editor.

It has 3 variables,

tiles: a list of all the normal tiles sets
autoTiles: a list of all the auto-tiles sets
plops: a list of all the plop sets

Each set then contain their own lists of tiles and plops.

= end =