Making Custom Maps for *Ultra Hat Dimension*

Guide ver. 1.3, updated December 1, 2015

Questions?
Contact @crowbeak on Twitter

Table of Contents

Introduction

Greetings and thank you for your interest in making custom maps for *Ultra Hat Dimension*! :D We're glad that you like the game enough to look into doing this and would love to see what you make. An easy way to contact us is via Twitter at @crowbeak, @enichan, or @yuzukimasu.

I, the guide writer, am Lena LeRay, the level designer for *Ultra Hat Dimension*. If anything is unclear or unexplained, please contact me @crowbeak or at press@kitsunegames.com.

This guide assumes you're already basically familiar with how the mechanics of *Ultra Hat Dimension* work, though some things are covered in detail herein.

We do not at this time have an in-game editor; all maps were made using the awesome and free Tiled map editor, which can be obtained at http://www.mapeditor.org.

For downloading and sharing custom maps please visit http://kitsunegames.com/uhd/maps/.

How to Use this Guide

Any text that must be input exactly, with care taken for things like uppercase letters and spaces, has a grey background. This is to avoid people putting quotation marks in places they don't belong. Yarr!

I have plans to supplement this guide with videos in the near future.

The most recent version of this guide will be available at http://www.kitsunegames.com/uhd/.

Using the Tiled Map Editor

Tiled can be downloaded for free from http://www.mapeditor.org/. It's available for Windows, Mac, and Linux. Although *Ultra Hat Dimension* is Windows only, you can totally construct maps on one of the other OSes and drop the files into the game later. In fact, most of the maps made for the game were constructed on a Mac.

Tiled is a pretty straightforward program. When you create a new map, it will ask you for dimensions in number of tiles and how many pixels tall and wide the tiles need to be. Make the tiles 16 x 16. You'll then need to import the tilemap image, tile.png, which can be found in *Ultra Hat Dimension*'s Maps directory.

IMPORTANT: The tile.png file you use for import *must be in the same directory where you **first** save the map file*[1] or else *Ultra Hat Dimension* will crash when it tries to load your map. If you forget (and I still do sometimes, 70+ maps later), this can be fixed (instructions below), but it sucks to deal with. I highly recommend putting a copy of tile.png in the directory where you plan to save WIP maps.

Once you've done all that, you just paint tiles on to the map until it's awesome.

Key features of Tiled that are important to understand for use with *Ultra Hat Dimension* are:

- Layers
- Objects
- File Formats

Layers

Tiled supports three kinds of layers: tile layers, object layers, and image layers. *Ultra Hat Dimension* only uses the first two kinds.

In the upper right-hand corner of the Tiled window is a list of layers; a new map in Tiled has only a single tile layer by default. Layers can be added, deleted, and rearranged in all the ways you might expect if you've ever used a modern paint program more sophisticated than the venerable MS Paint.

Objects

Tiled allows the creation of objects on special object layers. The editor supports objects of all shapes and sizes, but *Ultra Hat Dimension* does not. You'll want to go into the View menu in Tiled and make sure Snap to Grid is enabled. Use the rectangle object creation tool and make your objects take up exactly one tile.

[1] After the initial save, you can move the file around all you want. Tiled map editor will complain if you don't have a tile.png in the same directory when you load the file, but you can just close it, fix the problem, and reload the map without any further problems. What's happening is that Tiled saves the *relative* path if the files are in different directories and *Ultra Hat Dimension* won't process that. The game will crash if you try to load the file.

Map File Formats

The default file format for Tiled is *.tmx, but it also supports maps in *.json format. *Ultra Hat Dimension* only accepts the latter. The default file format allows Tiled to do some things the JSON format doesn't, but *Ultra Hat Dimension* maps don't need those features. I recommend just working with straight JSON files for ease of iterative testing.

Since JSON is just a text-to-objects parsing format, you could totally make all your maps by hand in a text editor. (Yuck.) You can also modify the JSON files directly to tweak things.

Fixing Broken Tile Image Paths

If you did not initially save your map in the same directory where the imported tile.png file lives, the game will crash when it attempts to load the map. If the map and the tile image are not in the same directory, it saves the tile image file name with a relative path and the game cannot handle that.

This can be fixed by opening up the yourmap.json file in a text editor and manually editing out the relative file path. Any plain text editor will work. Near the bottom of the file, you should find something like the following:

```
"tilesets":[
      {
       "firstgid":1,
       "image":" ..\/tile.png",
       "imageheight":480,
       "imagewidth":480,
       "margin":0,
       "name":"tile",
       "properties":
         {

         },
       "spacing":0,
       "tilecount":900,
       "tileheight":16,
       "tilewidth":16
      }],
```

You need to correct the red line to the following:

```
       "image":"tile.png",
```

And that’s it. Problem solved.

Making Layers

In *Ultra Hat Dimension* maps you *must* have at least two layers, one tile layer named canttouchthis[2] and one object layer named things. Anything in the canttouchthis layer will block player and Spluff movement; in the game, I used it for walls, bushes, and the like. The things layer is where you add things like the player character, the door, the key, Spluffs, hats, and switches. (These are covered below, in the objects section.)

Practically speaking, you will need *at least* one more tile layer for floors and other things you want the player to be able to see and yet be able to walk over. If you're using the game's default tile set, you'll also want one for the upper edges of the walls. I used six layers in all of the maps I made for the game.

Ultra Hat Dimension renders map layers in order from bottom to top, in whatever order they appear in the Tiled editor layer list. This allows things to overlap. The layers I used, from bottom to top, are: water, floor_lower, floor_upper, canttouchthis, things, and over_player.[3] This allowed me to add water channel wall decorations to floor_lower, for example, and to have tiles with transparent portions such as key pedestals, bushes, and bridges on top of other tiles so that there would be no strange black spots. The over_player floor was used to complete the wall borders, which were designed by our original artist to go… over… the player. (Surprise!)

I chose not to have secrets for players to find, but as an example: putting a piece of wall in a layer above the player instead of in canttouchthis would create a secret hole in the wall. The capability is there. Go for it if you wish.

[2] Yeup, MC Hammer reference. I'm a dweeb, you gotta deal with it. Special thanks to our lovely programmer Eniko for not objecting. (I suspect head shaking and/or eye rolling was involved.)
[3] The only names that matter are canttouchthis and things. All others were for my own reference.

Making Objects

Having an object layer named things is critical to your map not crashing when it tries to load. If there is no player object, for example, the game panics and things end poorly. On that note, your things layer must have a player object.

All other objects must have the correct name and, if required, a correct type field attribute. Some objects require extra fields to be added. Some objects support optional extra attributes.

When an object is created in Tiled, it doesn't initially have a name or type. With the object selected, these fields can be modified in the lower right-hand corner of the Tiled window. This is also where new attribute fields can be added and filled in.

Here's the full list of objects supported by *Ultra Hat Dimension.* Objects in red are required:

Object name	Type field values	Required extra fields	Optional extra fields
player	N/A	N/A	N/A
door	1, 2, 3, 4, or 5[4]	N/A	N/A
key	N/A	N/A	N/A
guest[5]	a, b, c, or d[6]	N/A	possessed
hat	a, b, c, or d	N/A	possessed
switch	1, 2, 3, 4, or 5	target	N/A
fab	[empty], a, b, c, or d[7]	N/A	N/A
jailedfab	N/A	N/A	N/A
guard	N/A	N/A	N/A
king	N/A	N/A	N/A
queen	N/A	N/A	N/A

The optional possessed field for guests and hats makes them glowy and alters those Spluffs' behavior. A possessed guest Spluff will ignore the player's hat, even if it matches the Spluff's type, and punch the player away; the player cannot ask them to move. Set the value of possessed to anything *except* false or 0 to activate possession. I used true.

[4] Numbers correlate to floor numbers and associated themes for both doors and switches. 1 is spring (pink and purple), 2 is summer (greeeeeeen), 3 is autumn (orange and yellow), 4 is winter (blues), 5 is drab (why Fab why). If you forget this, the door will still render, but have a 1 in 5 chance of being the wrong color.

[5] This is the basic Spluff. Fun fact: originally we talked about having different Spluffs with different behaviors. Guards, the king and queen, and jailed Fab made it in at the very end, but are only decorative.

[6] Letters correspond to specific hat types. a is ribbon/bow hat, b is star hat, c is wing hat, d is wizard hat. A given type will knock out the type of the next letter (a will knock out b) and knock back the type of the previous letter (a will knock back d). Type a/c and b/d interactions cause slapfights.

[7] If left empty, Fab will have his hat fully intact but no parts of the hat will glow. Values a, b, c, and d correspond to specific parts of his hat glowing. However, since the game was designed with those pieces getting punched off in a certain order, Fab also gets more beaten up with each iteration.

Switch Properties

The target field required by switches is used to specify which tiledefs it activates; in the base game, this is only used for toggling bridges on and off, but when combined with the magic of tiledefs[8] its potential uses are many in number. If you forget the target field, the game will not crash; instead, the switch will simply not render in the game.

To specify which bridges or other tiledefs a switch can toggle, fill the target field with a list of coordinates like one of the following examples: 7,3 , 12,4;3,6;20,20. A rectangular range can also be specified using the coordinates of the top left and bottom right tiles in the rectangle, like so: 1,5 - 6,8.

To find a the coordinates of a desired target space, hover your mouse over it in Tiled and check the lower left corner of the Tiled window. If you resize the map, coordinates may change, in which case they must be altered manually. (Sorry. Trust me, I know how much this sucks.)

Switches also have two optional properties, switchonce and sound.

If the value of switchonce is not false or 0 this will cause the switch to only function once.

If the sound property is present, the game will try to play the sound (plus .ogg extension[9]), crashing the game if the appropriate sound file is not present. If the property is left empty it will cause the switch to play no sound when toggled. If not present, the default switch sound is played.

[8] Actually magic. Eniko is a bonafide wizard. Certified by a totally legit authority.
[9] This is identical to how music is handled in the map properties.

Setting Map Properties

Tiled allows users to give maps additional properties to maps in much the same way they can do for objects. These can be found by selecting Map Properties in the Map menu and looking in the lower right hand corner of the editor and are added in the same way as object properties. *Ultra Hat Dimension* requires two additional map properties, name and music.

The name will be displayed in both the map selection screen and at the bottom of the screen when playing the level. For the name, *Ultra Hat Dimension* supports unicode, but the characters that will display are limited by the font. The font cannot be replaced at this time. Sorry.

If you make the name really, really long, the game will not crash, but it also won't be pretty. The game automatically adjusts font size in the map selection screen and it can become so small as to be unreadable. When displaying the name in the level itself, extremely long names just won't fit on the screen.

The music field specifies the background music that will loop during play. The only audio format supported by *Ultra Hat Dimension* is *.ogg, and the value that goes in the music field is the audio file name minus the extension. You can add custom music, but by default the music field may contain any one of the following values:

stage1
stage2
stage3
stage4
boss
gameend
intro
interlude
victory

Optional map properties are creator and thumbnail. The creator property allows you to add attribution to maps, which will be shown in the custom map selection screen. The thumbnail property allows you to set a custom thumbnail location by specifying x,y coordinates. These can be whole numbers as in 4,7 or decimals as in 1.5, 9.75.

If you poke around in the map files created for the base game, you might notice that they all have a map property called par. This field was added for a planned feature that we ran out of time to implement. The par field can therefore be safely ignored.

Turn Order and the Object List

At the bottom of the layer list in Tiled are three tabs labeled Mini-map, Objects, and Layers. The Layers tab shows the layer list, the Mini-map shows your entire map in tiny glory, and the Objects tab shows a list of all the objects in the map. This is super useful for making sure that all of your object names and types are correct and for finding specific objects in a sea of squares.

More importantly, the order in which guest objects appear in this list is directly linked to how punching is processed in the game. Every turn, things happen in the following order:

1. The player initiates movement.
2. If the target square has a hat in it, the player picks it up.[10]
3. If the target square has a Spluff in it and the player's hat matches (which it presumeably does because the player is already standing next to it), the Spluff will be asked to move; player movement is cancelled and they remain in the square.
4. If a Spluff was asked to move, the Spluff will move until it runs into something it can't pass through (a tile in the canttouchthis layer, another Spluff that isn't knocked out, or a river) and then check adjacent squares for things to punch.
5. All other Spluffs are processed *in the order in which they appear in the object list.*
    a. Is the Spluff on solid ground?
    b. If no, if it's in a river, it flows until it hits something it can't pass through or ends on solid ground.
    c. It punches anything around it.

This means that if you want to set up punch chains that send Bea (or other Spluffs) around the level automatically, you'll have to make sure your Spluffs are ordered correctly.

[10] BEWARE! If the target square contains both a hat object and a guest object, the player will pick up the hat before asking the Spluff to move. Counter-intuitive and potentially problematic.

Custom Assets

You can include custom assets for your maps by bundling the map(s) and the associated custom asset(s) into a zip file. Custom assets and default assets can be mixed, so if you want to make your own graphics but rely on the default music, you can. You can also have custom tiles but rely on the default player and Spluff graphics, etcetera.

The only supported audio type is OGG.

Supported image files are BMP, GIF (not animated), JPEG, PNG, and TIFF. PNG is the recommended format.

Tileset images can have any name they want as long as the map points to the right file. Failure to find the tileset image when loading the map will cause *Ultra Hat Dimension* to crash.

Other assets (images and sounds) must be named identically to the originals to replace them. Image replacements should be the same size as the originals. They *must* be at least as big as the originals, though there is no point in making anything except tilesets bigger.

Since the tiles are loaded when the map is loaded, you can have multiple tilesets in your zip archives. However, this is not recommended because of tiledef support. Tile IDs are global, not limited to a specific tile set, and you probably don't want your lava tiledef to trigger when people walk onto flowers[11]. It's better to make a single, large tileset image.

[11] Probably. You monster.

The Magic of Tiledefs

Tiledefs are bits of code magic that allow for things seen in the game such as animated water tiles, Spluffs flowing downstream, and switches toggling bridges. This works by applying special properties to specific tiles, using their tile IDs.

Resident wizard Eniko set up tiledefs for *Ultra Hat Dimension* which make bridges toggle, animate water and torches, and make Spluffs float downstream. The tiledef system is actually more flexible than that, especially since Eniko added a few things that aren't used in *Ultra Hat Dimension*. See below for information on how to make your own tiledefs.

Using Default Tiledefs

Bridges are easy. They can start extended or retracted, depending on which tile you lay down, and if you set a switch to target them, the switch will toggle them when pressed.

Water and torches are also easy to use. Just use the stream-end grate tiles and torch tiles that are *clustered with the rest of their associated color scheme*. For water, use the cluster of five tiles at the bottom center of the tilemap image. Yes, the ones with the really blatant arrows and the still water image right above them; it looks like a thumbs-up, kinda.

If you look at the lower right-hand corner of the tilemap image, you'll see all the frames of animation for those things. *Don't use those when mapping your maps!* They are animation frames referred to by the tiledef specification. The game will animate them automagically.[12]

[12] Because Eniko is awesome. She programmed all this cool stuff and also made the animations. Certified. Wizard.

Creating Custom Tiledefs

This is technical and best understood if you have some understanding of code.

The 'tiledefs.json' file is just a json file structured like this:

```
{
   "tiledefs": [
      <tiledef>,
      <tiledef>,
      <tiledef>
   ]
}
```

A tiledef is an object that looks like this:

```
{
   "id": <tile id>,
   "properties": {
      <property1>,
      <property2>,
      <property3>
   },
   "frames": [
      <frame1>,
      <frame2>,
      <frame3>
   ]
}
```

The <tile id> is the id that Tiled shows in the lower right-hand corner when you click on the tile in a tileset. If you have multiple tilesets, this won't necessarily be accurate because the game uses the global tile id, which Tiled doesn't show. We recommend not having multiple tilesets in a single level.

Properties and frames are both optional. The former is used for functional changes to be enacted by the tiledef; the latter is for animating tiles.

Properties

A property is always a key/value pair, where the key provides the type and the value provides additional information. Valid types and values are listed below:

linked	This tile is linked to another tile. This tile will be changed to its linked tile when a switch that targets a position on the map containing this tile is activated. A tile that links to a tile which links back to the first will cause a tile that flips between two states, but this does not need to be the case. For example, tile X could link to tile Y, which in turn could link to tile Z, etcetera. If Z then links back to X or Y, the sequence would loop either in part or in full, if Z does not link to any tiles then the sequence ends and the tile would become permanently unswitchable.
	The value must be a numbered tile id.
sound	When an entity moves onto this tile the game plays the specified sound.
	The value for this tiledef must be a sound file either found in the base game's Content folder or in the custom map's zip file, in ogg format, minus the extension. Example: "sound": "kagi1"
music	When an entity moves onto this tile the game plays the specified music, stopping any currently playing music.
	The value for this tiledef must be a sound file either found in the base game's Content folder or in the custom map's zip file, in ogg format, minus the extension. Example: "music": "stage1"
reset	When the player moves onto this tile the level resets immediately.
	If the value is not a string (null, true, false, or a numerical value) the default "hmmm" reset sound is played, otherwise the value for this tiledef must be a sound file either found in the base game's Content folder or in the custom map's zip file, in ogg format, minus the extension, which will play instead. Example: "reset": "hat2"
blocks	This tile blocks movement.
	Accepted values are "all", "guest" and "player", any other value will default to all, this specifies which type of entities have their movement blocked by this tile. Example: "blocks": "all"
water	This tile is water, causing the entity (player or guest) to be rendered differently
	If the value is a valid HTML style color the line rendered at the bottom of the sprite will be this color, otherwise it will default to the default water color. Format is #RRGGBBAA with the alpha component being optional. Example: "water": "#FF0000" would render a red line.

push	This tile pushes some or all entities that stand on it into a certain direction.
	The value must start with a direction: "left", "up", "right", "down". An optional specification "all", "guest" or "player" can be added after a comma to indicate which type of entity is pushed by the tile. If the second value is invalid or missing it defaults to all. Example: "push": "left, guest"
bridge	This tile functions as a bridge. If a bridge tile is on the same tile as a push tile then any entities that move onto that tile will not be pushed. If a bridge tile is on the same tile as a blocking tile then entities will not be blocked (only when blocking through tiledefs! tiles on the "canttouchthis" layer will always block). If a bridge tile is on the same tile as a reset tile and the player moves onto it the game will not reset.
	The value must be either true (this tile is a bridge) or false (this tile is not a bridge). Example: "bridge": true

Frames

Frames is used for animating tiles in place and is a list of frame objects structured like so:

```
{
   "id": <tile id>,
   "duration": <duration>
}
```

The tile id here is the same as explained above. Duration specifies all frames' duration in seconds and can be either a simple value (for example 0.2 would last 0.2 seconds) or a random value in a certain range (for example 0.2-0.4 would last anywhere between 0.2 and 0.4 seconds, determined randomly).

Example tiledef.json

```
{
  "tiledefs": [
    {
      "id": 3,
      "properties": {
        "water": "#FF0000",
        "sound": "pow"
      },
      "frames": [
        {
          "id": 3,
          "duration": 0.2
        },
        {
          "id": 4,
          "duration": 0.2
        },
        {
          "id": 5,
          "duration": 0.2
        }
      ]
    },
    {
      "id": 6,
      "properties": {
        "bridge": true,
        "linked": 7
      }
    },
    {
      "id": 7,
      "properties": {
        "bridge": false,
        "linked": 6
      }
    }
  ]
}
```

Troubleshooting

If your map is crashing when the game attempts to load, check for the following issues:

- Your tile image path does not contain garbage.
- All objects have their required properties filled in.

If your switches are not appearing, make sure they have a valid target field.

If you've set up a punch chain and it's not working the way you want, check the involved Spluffs' order in the object list.

If you've included custom art or music assets, consider also checking for the following issues:

- The sound files are *.ogg files.
- The object properties pointing to sound files are filename minus extension.
- Art replacement assets are named identically to the defaults included with the game, including the same extension.
- Art replacement assets are at least as big as the originals (preferably the same size).
- Art replacement assets are filename with extension.
- All custom assets are in the root of the zip archive, not in subfolders.

Where to Put Your Spiffy Maps

Since we have both zipped and installer versions of *Ultra Hat Dimension*, there are two places where custom maps can go. You can use either or a combination of both; the game loads maps from both places when the game is run.

The first place you can put them is the Maps directory inside the *Ultra Hat Dimension* directory. If you are using the zipped version of *Ultra Hat Dimension*, that is easy to do.

Windows doesn't allow users to just drop files into Program Files and its subdirectories, though, so the other place where map files can live is the %APPDATA%\Ultra Hat Dimension\Maps folder. If you used the installer version of *Ultra Hat Dimension*, that's where you should put them. If you can't find this directory, use the shortcut to it in the Kitsune Games\Ultra Hat Dimension\Maps folder.

Bundling Your Spiffy Maps into Spiffy Packs

If you want to create your own mini-campaign… you can! If you make a zip archive with all your maps in it (and specifically in the base folder, no subfolders), players can drop the archive straight into their custom maps directory without having to unzip anything. Any maps in the zip file will be displayed in alphabetical order.

Since maps don't need to be unzipped to be used by the game, this is also the best way to distribute maps with custom assets. All maps and assets must be in the base folder of the zip, not in subfolders.

A special file info.json can be added to name the map set and creator, in this format:

```
{
   "name": "Example Map Set",
   "creator": "Kitsune Games"
}
```

For single maps the `name` and `creator` fields in the map's properties are used.

Where to Find Examples

We've left a couple of examples in the custom maps directory for you, but you can also go into the game directory itself and look at any of the maps we made for the game. It's all JSON files. Have fun.