

Tiled Plugin

A plugin by Dr. Yami and Archeia

Introduction

- Tired of RPG Maker MV's Map Editor?
- Do you want to map the XP way but more?
- Tired of Parallax Mapping?
- Want to do round corners?
- Want to create a map with basically unlimited layers?

Well, now all those worries are gone! Instead, let's just use the awesome map editor, Tiled! Free, easy to use and very flexible Map Editor. This is one of our reveals for RMMV's release but due to unforeseen circumstances, we were unable to showcase this really awesome plugin in RPG Maker Channel.

-- Archeia and Dr. Yami

Tiled is a separate application developed by Bjorn, and you can find it here:

<http://www.mapeditor.org/>

Usage

Just put this script as high as possible, preferably before plugins that extend functionality of maps and rendering thereof, but after any plugin that rewrites map handling. Ideally, you'd only have to use this plugin for any of your mapping needs.

Note that the Tiled plugin handles the following:

- Map layouts and rendering
- Regions
- Collisions
- Parallax images

Note that this guide won't explain how to use Tiled itself, this will only go over the plugin specific features.

Starting your project

After you've added the Tiled plugin you may want to start creating your Tiled maps. Make sure you make these maps in your game folder, so that any direct references to images will work right off the box.

The first thing you'll need to do is to create a map inside RPG Maker MV. By default, you'll already have a map with map ID 1, so when you start off, you can just use that map. Now, create a new map in Tiled. Make sure its orientation is Orthogonal, and the map is saved as a comma separated value (CSV).

Next, save your map as a json file. Call it Map, followed by the ID, but not zero padded. For example, if your map ID is 1, save your Tiled map as Map1.json. Make sure you directly save it in the folder you defined in your plugin settings (Maps Location). And with that, you've set up your first map in Tiled.

Infinite maps vs. fixed size maps

Tiled has two types of maps, infinite maps and fixed size maps. They are essentially similar in use, with one difference, which is that infinite maps don't have a fixed size. In the Tiled plugin, both maps are treated the same, though, with infinite maps having their size determined by the sizes defined inside the Tiled map format.

This means that infinite maps may appear smaller or cut off than what they are supposed to look like. To fix this, you could preset the sizes by manually setting the size. Your map will remain an infinite map, but you'll now have a bit more flexibility with how the map is shaped in the final product.

It's still recommended to convert infinite maps to fixed size maps, as the pre-processing phase will be skipped, allowing for faster map loading.

Layers

Like stated before, with this plugin you can use practically unlimited layers. By default, a layer has no z-index, which means it'll always be at the bottom, or stacked on top of other layers and objects with no z-index.

Tilesets

By default, RPG Maker MV tilesets won't work in Tiled, since autotiles aren't implemented. However, there is a tool that can convert RPG Maker MV tilesets so that it can be used in Tiled, called Remex.

<https://app.assembla.com/spaces/rpg-maker-to-tiled-suite/subversion/source>

When importing tilesets, you have the option of embedding the tileset data inside the map itself, or saving it in a separate file. Make sure that when you do the latter, you'll have to save it as a json file, and you'll have to put it in the folder specified in the plugin configuration, under Tilesets Location.

Events

Setting up events is easy, although it does require you to actually still use RPG Maker MV, as you'll be setting up your events here. Like maps, each event has its own ID. To retrieve its ID, you'll have to select your event, and in the bottom right corner of the RPG Maker MV editor, you'll see the event ID as well as its name, for example, 001:EV001.

Next, you'll need to create a new object in Tiled. Create a new Object layer, and create a new rectangular object. You could in theory use a different shape, but since events generally are rectangular, you want to make sure your object is the same shape as a tile. Also, if Constrain Events to Grid is set to true, the objects will automatically snap to the grid, so, if you don't want your event to appear somewhere you didn't intend it to be, make sure you snap to grid.

Now, to make it all work, all you need to do is give this object a property eventId and set the event ID as the value. This will move the event to the proper map location.

Vehicles

You can add vehicles in the same way as events, except you don't have to place the vehicle in your map. This is because vehicles already are on every map.

To place a vehicle in your map, use the object property vehicle, and set as the value the name of the vehicle you want to add. You can use "boat", "ship" or "airship", or any custom made vehicle you have made through plugins.

Extra notes on parallax images

While the Tiled plugin has its own handles for parallax images, you can still use RPG Maker MV's default parallax image functionality, as that will not be overwritten. This is to give the creators options, in case the Tiled functionalities are too confusing.

Properties

As explained earlier, several elements in Tiled can have properties that will interface with the Tiled Plugin.

Tile layer properties

zIndex

The layer will have z-index equal to the property's value.

collision

The layer will be a collision mask layer. Use one of these value:

- full - Normal collision (1 full-tile)
- arrow - Arrow collision
- up-left - Half-tile collision up-left quarter
- up-right - Half-tile collision up-right quarter
- down-left - Half-tile collision down-left quarter
- down-right - Half-tile collision down-right quarter
- tiles - Collision is determined by the tileset

arrowImpassable

If the layer is an arrow collision mask layer, it will make one direction be impassable. Value can be up, down, left or right.

regionId

Mark the layer as region layer, the layer ID will be the value of property. If set to -1, it will use the tileset to determine the region ID.

priority

Mark the layer as priority layer, allows it goes above player when player is behind, below player when player is in front of. Value should be greater than 0, zIndex should be the same as player z-index.

Note that this also determines the drawing priority, e.g. layers with the same z-index and y-position would then be sorted by their priority.

level

Mark the layer on different a level, use for multiple levels map (for example a bridge). Default level is 0. Use this for collision and regionId.

toLevel

The tiles on this layer will transfer the player to another level.

tileFlags

Set this to true to enable tile flags in the tileset. Set this to "hide" if you aren't going to draw this layer.

hideOnLevel

Hide the layer when on a certain level.

showOnLevel

Show the layer when on a certain level.

hideOnRegion

Hide the layer when on a certain region.

hideOnRegions

Hide the layer when on a certain region. This takes an array of possible regions, and the player only needs to be on one of the layers. Each region is comma separated.

hideOnAnyRegions

Hide the layer when on a certain region. This functions the same as hideOnRegions, except it will take all regions on that tile instead of just the top visible region.

hideOnSwitch

Hide the layer when a certain switch is set.

showOnSwitch

Show the layer when a certain switch is set.

transition

This will enable the layer to transition from a shown state to a hidden state. Set the value as the duration in ticks.

minimumOpacity

The minimum opacity the layer should fade out to.

Tileset properties

ignoreLoading

Ignores the image, and doesn't load nor render it. Good if you want to mark certain areas in Tiled for development purposes or mapping notes, but don't want them to appear in the game.

Tile properties

regionId

Mark the tile as region, the tile's region ID will be the value of property

collision

Mark the tile as having normal collision (1 full-tile)

collisionUpLeft

Mark the tile as having half-tile collision up-left quarter

collisionUpRight

Mark the tile as having half-tile collision up-right quarter

collisionDownLeft

Mark the tile as having half-tile collision down-left quarter

collisionDownRight

Mark the tile as having half-tile collision down-right quarter

collisionUpLeft, collisionUpRight, collisionDownLeft and collisionDownRight can be combined to create a custom collision.

arrowImpassableLeft

Make the left direction impassable

arrowImpassableUp

Make the up direction impassable

arrowImpassableRight

Make the right direction impassable

arrowImpassableDown

Make the down direction impassable

arrowImpassableLeft, arrowImpassableUp, arrowImpassableRight and arrowImpassableDown can be combined to create a custom arrow passability.

flagIsBoat

The tile is passable by boat

flagIsShip

The tile is passable by ship

flagIsAirship

The tile is landable by airship

flagIsLadder

The tile is a ladder

flagIsBush

The tile is a bush or has special bush rendering

flagIsCounter

The tile is a counter (can interact through this tile with another event)

flagIsDamage

The tile is a damage tile (player gets damaged when stepped on)

flagIsIce

The tile is slippery

Object properties

eventId

The event ID that should be placed at this position.

vehicle

The vehicle that should be placed at this position.

Image properties

ignoreLoading

Ignores the image, and doesn't load nor render it. Good if you want to mark certain areas in Tiled for development purposes or mapping notes, but don't want them to appear in the game.

zIndex

The image will have z-index equal to the property's value.

repeatX

Whether it has to repeat horizontally or not.

repeatY Whether it has to repeat vertically or not.

deltaX

The horizontal movement when the camera moves.

deltaY

The vertical movement when the camera moves.

autoX

Setting this will automatically move the image, depending on the value, in the horizontal direction.

autoY

Setting this will automatically move the image, depending on the value, in the vertical direction.

hue

The hue of the image.

hideOnLevel

Hide the layer when on a certain level.

showOnLevel

Show the layer when on a certain level.

hideOnRegion

Hide the layer when on a certain region.

hideOnRegions

Hide the layer when on a certain region. This takes an array of possible regions, and the player only needs to be on one of the layers. Each region is comma separated.

hideOnAnyRegions

Hide the layer when on a certain region. This functions the same as hideOnRegions, except it will take all regions on that tile instead of just the top visible region.

hideOnSwitch

Hide the layer when a certain switch is set.

showOnSwitch

Show the layer when a certain switch is set.

transition

This will enable the layer to transition from a shown state to a hidden state. Set the value as the duration in ticks.

minimumOpacity

The minimum opacity the layer should fade out to.

Development hooks

For the Tiled Plugin, several hooks have been created for plugin developers, so that they could create their own plugin for Tiled without having to overwrite functionality.

```
TiledManager.addListener(objectName, event, callback, recursive = true)
```

Adds a listener for certain events.

objectName

The name of the object this listener is meant for, for example, `Game_CharacterBase` or `Game_Player`.

event

The name of the event it should listen to.

callback

The callback function. This callback function takes exactly one argument, as the listener will pass an object to this callback.

recursive

Whether the event callback will be triggered recursively. If an object is derived from another object, like `Game_Player` being derived from `Game_CharacterBase`, by default it will also call the callbacks from that object. When set to false, it will skip the current callback if it's not directly applied to that object.

```
TiledManager.triggerListener(object, event, options = {})
```

This triggers an event for a certain object.

object

Generally the keyword "this", but you can set any object for which the callbacks should be called.

event

The name of the event. that should be called.

options

An object that contains properties that can be passed on to the callback.

```
TiledManager.addHideFunction(id, callback, ignore = [])
```

Adds a hide function, a function that hides a layer or image based on certain rules.

id

The ID of the hide function. This will also be used as a property check for Tiled maps itself. If there isn't a property of the same name as the ID, it will not execute the callback.

callback

The callback function that should be run. This should return true if a layer or image has to be hidden, and false if it should remain visible.

ignore

This is mainly used during calculations, and can be left empty. Essentially you can add groups that should ignore this function. These groups are:

- regions
- collisions
- levelChanges
- tileFlags

The only thing it does is making sure functions and methods that belong to a certain group will ignore this callback.

```
TiledManager.checkLayerHidden(layerData, ignore = [])
```

This will check if a certain layer is hidden.

layerData

The Tiled layer data. Note that this isn't the properties data, but the entire layer data.

ignore

You can specify here which hide functions it should ignore, as specified in TiledManager.addHideFunction.

```
TiledManager.hasHideProperties(layerData)
```

Checks whether a layer has hide properties.

layerData

The Tiled layer data. Note that this isn't the properties data, but the entire layer data.

```
TiledManager.addFlag(...flagIds)
```

Adds a flag or multiple flags.

flagIds

The flag ID you want to add. You can just set multiple of them as multiple arguments. The flag IDs will be used to check up which bit you need to check, as well as in the tile property. When setting a tile property using this flag ID, prepend it with "flagIds" and capitalize the first letter. For example, if you have a flag ID "monster", you'll use the property

"flagIsMonster".

```
TiledManager.getFlag(flagId)
```

Get the numerical ID of a certain flag.

flagId

The flag ID.

```
TiledManager.getFlagNames()
```

Get a list of all flags.

```
TiledManager.getFlagLocation(flagId)
```

Get an array with the values group and bit. The group is the group in which the flag bit resides, and the bit is the bit number of that group.

flagId

The flag ID.

```
TiledManager.createVehicle(vehicleName, vehicleData = false)
```

vehicleName

The name of the vehicle.

vehicleData

An object containing the vehicle data. When false, it will take default values, otherwise, it will take an object. This object has essentially the same structure as the vehicles inside the System.json file. On top of that, there are a few extra properties.

```
moveSpeed
```

The move speed of the vehicle.

```
direction
```

The initial direction of the vehicle.

```
tileFlag
```

The tile flag it looks for when looking for passability.

```
hasCollision
```

Whether the vehicle has collision. When false, you'll have to stand on top of it to enter.

`resetDirection`

Whether the vehicle has to reset its direction after you exit it.

Objects and events

These are the objects and their events. Note that events are inherited by underlying objects, for example, `Game_Character` inherits all events from `Game_CharacterBase`.

All elements will pass an object with several properties, which you can use in your own events.

`Game_CharacterBase`

stopmovement

Triggers when movement is stopped.

`direction`

The direction the character is looking in.

slipperyfloor

Triggers when stepping on a slippery floor tile.

`direction`

The direction the character is looking in.

`Game_Map`

changelevel

Triggers when changing layer levels.

`oldLevel`

The old level.

`newLevel`

The new level.