Based off Verison 1.08

Table of Contents:

Movement

- Colliders
- Tile Boxes and Region Boxes
- Collision Maps and Region Maps
- Passability Levels
- >Moving Characters
- Advanced features

Colliders

There are two types of colliders; Box Colliders and Circle Colliders. You can config the colliders width, height and offsets. For Circle Colliders you won't set a radius, instead you configure it if you are making a box and circle will be created inside the box.

Setting up Colliders

Colliders are created inside the Players notebox or as a comment inside an Events page. Events colliders depends it's page, so you may need to make the collider on all pages.

Collider (Note Tag)

```
<collider=type,width,height,ox,oy>
```

Type: Set to box or circle.
Width: Set to the width of the box / X Diameter of the circle.
Height: Set to the height of the box / Y Diameter of the circle.
OX: Set to the x offset of the collider relative to it's px value.
OY: Set to the y offset of the collider relative to it's py value.

You can also set up a different collider depending on the direction the character is facing.

```
<collider>

5: type, width, height, ox, oy

X: type, width, height, ox, oy

</collider>
```

Where 5 is the default box if a box isn't set for a direction.
And X is the box for that direction.
Direction can be 2, 4, 6 or 8.

Tile Boxes and Region Boxes

Tile boxes are automatically created. Their sizes will also adjust depending on your **Tile Size** settings. Tile boxes merge will all the tile layers. So if the bottom layer is impassable but the top layer is passable, the whole tile will be impassable. To get around this we can use Region Boxes. Region Boxes take priority over Tile Boxes, therefore we can create a passable region box and place it over an impassable tile to make it passable.

In order to be able to use Region Boxes, you will need to enable the **Use Region Boxes** setting. once it's enabled you will need to create a json file called "RegionBoxes.json" inside the **JSON Folder** you set in the settings.

JSON template (JSON)

```
{

"REGION ID 1": [

   {"width": w, "height": h, "ox": ox value, "oy": oy value}

      ],

"REGION ID 2": [

   {"width": w, "height": h, "ox": value, "oy": value},

   {"width": w, "height": h, "ox": value, "oy": value}

      ]

}
```

Region ID: The region that will use this box. Be careful with the commas (,) place them after } or] only if it's not the last one in the list!

Example JSON File.

Collision Maps and Region Maps

If you don't want your tile collisions to be all boxes and have unique shapes, you can use Collision Maps. Collision Maps will overlay Tile Boxes, so if you don't want conflicting Tile Boxes, make the whole map passable by eithre editing the tiles passability in the Tileset Editor, or use a passable Region. Region Maps are an extra tool for developers and for Region Maps may be used in addons.

Creating Collision Maps / Region Maps

You can create the maps in any image editor you want. For Collisions Maps use the color you set for **Collisions** in the settings to indicate collisions. You can use any color you want in Region Maps. Place these images in the folder you set in the settings for **Collision Map Folder** and **Region Map Folder**

Enabling Collision Maps (Notetag)

```
<cm=filename>
```

Enabling Region Maps (Notetag)

```
<rm=filename>
```

filename: Set this to the name of the image you want to load.
* These notes should be placed inside the Maps Notes found in the Map Properties.

Passability Levels

Passability levels are a new feature which sets wither a character can walk over water or deep water tiles.

Levels:

- **0 -** Default, Can only move on passable tiles
- **1 -** Boat, Can only move on water 1 tiles
- **2 -** Ship, Can only move on water 1 and water 2 tiles
- **3 -** NEW, Can move on passable tiles and water 1 tiles
- **4 -** NEW, Can move on passable tiles, water 1 and water 2 tiles

Set default Passability Level for Event(Comment)

```
<pl=X>
```

X: Change X to the passability level you want to set it to.
* Resets on page change.

Changing a Characters Passability Level (Script Call)

```
$gamePlayer.setPassability(X);
```

For Events:

```
$gameMap.event(ID).setPassability(X);
```

ID: The ID of the event.
X: Change X to the passability level you want to set it to.

Check a Characters Passability Level (Script Call)

```
$gamePlayer.passabilityLevel();
```

For Events:

```
$gameMap.event(ID).passabilityLevel();
```

ID: The ID of the event.
These will return the value of their passability level.

Moving Characters

Moving characters can be pretty tidious when using pixel movement. So there are a few new functions to make it easier.

Script calls for Move Routes

The following should be placed inside the "script..." command in Move Routes.
QMove (Quasi Move) Moves the character X amount of distance, ignores **Off Grid** setting

```
qmove(direction, amount, multiplicity)
```

direction - Which direction the movement should be.
amount - How many times should the player move.
multiplicity - Multiplies amount for easier calculations.

MMove (Multiple Move)
Moves the character X amount of distance, stays on Grid.

```
mmove(direction, amount, multiplicity)
```

direction - Which direction the movement should be.
amount - How many times should the player move.
multiplicity - Multiplies amount for easier calculations.

Initializing Events OX and OY

You can fine tune your events starting location, so they will stand at the pixels you want them to.
Setting initial Offsets (Comments)

```
<ox=X>
```

X: Change X to the inital X offset.

```
<oy=Y>
```

Y: Change Y to the inital Y offset.

* Offsets resets on page change.

Jumping

There are two new functions for jumping. A Pixel based jump and a jump forward.

Pixel Jump (Script Call)

```
$gamePlayer.pixelJump(distanceX, distanceY)
```

For Events:

```
$gameMap.event(ID).pixelJump(distanceX, distanceY)
```

ID: The ID of the event.

Jump Foward (Script Call)

```
$gamePlayer.jumpForward(direction)
```

For Events:

```
$gameMap.event(ID).jumpForward(direction)
```

ID: The ID of the event.
direction: Direction can be 2, 4, 6 or 8.

Advanced features

Advanced features are only for users that are comfortable with scripting. All these features may require some knowledge for Javascript to use to it's full potiental.

Get tile flags

```
$gameMap.flagsAt(x, y);
```

x and y default to players x and y
Works best when grid is equal to tile size. The results are logged in the console.

Get color from region map

```
$gameMap.getPixelRegion(x, y);
```

x and y default to players center location.
return value is a string of the hex color.

Create a collider

```
var myBoxCollider = new QuasiMovement.Box_Collider(w, h, ox, oy, shiftY);
```

For circle:

```
var myCircleCollider = new QuasiMovement.Circle_Collider(w, h, ox, oy, shiftY);
```

x and y default to players center location.
return value is a string of the hex color.

Moving Colliders

```
myCollider.moveto(x, y);
```

Set X and Y in pixel terms.
Also use the variable that you used to make the collider.

Showing Custom collider on map

```
SceneManager._scene.addTempCollider(collider, duration);
```

(Only works if you're in Scene_Map!)
Set collider to the collider object
Set duration to the duration it will display

Get characters that overlap with a collider

```
$gameMap.getCharactersAt(collider, ignore);
```

Collider needs to be a Collider object
Ignore is a function
Returns an Array of characters it overlays.
(Search this plugin for an example usage if needed)

Get map tiles that overlap with collider

```
$gameMap.getTileBoxesAt(collider);
```

Collider needs to be a Collider object
Returns an Array of tilesboxes it overlays.
You will need to manually filter this array because there is no ignore parameter, so it returns all tiles.
(Search this plugin for an example usage if needed)