A semi-automatic mapper for TOAW IV in QGIS

Markus Gattringer

December 10, 2020

Contents

Part I
Preamble

1 Requirements

1.1 QGIS

The first thing that is required is QGIS, a free and open geographic information system (GIS). QGIS is basically the lower-key alternative to ArcGIS. With QGIS, amends have to be made regarding capability, but at the advantage of not having to pay a fortune for a GIS software. Despite this downside, QGIS is still a very powerful tool, and more than sufficient for the task at hand.

QGIS is available here: `https://qgis.org/en/site/`. Any recent version should do; version 3.14 has native support for vector tiles, which might come handy. This guide was created using 3.16, except for the terrain creation, as currently automatic conversion between CRS is not handled by the raster analysis algorithm. Additional plugins are not needed; the base installation works fine.

QGIS can be crash prone, depending upon the task, system configuration, and the type and amount of layers loaded. This includes reproducible system freezes for certain grid sizes. So, *caveat emptor*.

Familiarization with the basic functions of the program is recommended. While the process described in this document can be mostly followed to the letter, there are preparatory processes that are not covered here. Additionally, in subsection 3.2 manual analysis of a layer needs to be performed.

1.2 DEM

To classify the terrain, a digital elevation model (DEM) is required. DEM usually comes in the form of Geo TIFF lossless compressed gray scale raster files. DEM data is widely available: there are global 90 m-resolution datasets available, regional datasets with resolution in the 30 m-area (*e. g.* European datasets at `https://land.copernicus.eu/imagery-in-situ/eu-dem`) and local datasets generated from LIDAR data with resolution below 10 m (*e. g.* an Austrian dataset at `https://www.data.gv.at/katalog/dataset/d88a1246-9684-480b-a480-ff63286b35b7`). Those sets may or may not be available for free.

It may be necessary to combine and rescale several DEM tiles to achieve complete coverage. The guide will assume that an appropriate DEM file is used.

1.3 Basemap

A vector basemap, generally consisting of paths denoting line features like roads and rivers, and polygons denoting area features like forests and bodies of water, is the second requirement. GIS data handled by the authorities is generally in such a format.

Vector base data is generally not available as easily as DEM data; and if it is, one usually needs to purchase it. This project was started due to the easy and free availability of the official vector base map for Austria at `https://www.data.gv.at/katalog/dataset/b694010f-992a-4d8e-b4ab-b20d0f037ff0`. Though, in principle, one can always inquire with the respective authorities — although that is problematic in itself, since geographic information is generally handled at the level of provinces, rather than states.

A feasible alternative would be the use of OpenStreetMap. There is a preinstalled plugin (meaning that it needs to be activated) that lets one parse OSM data from within QGIS. While the OSM data has its problems (like gaps and a redundant and overly detailed structure), it is often the only possibility of obtaining data.

As ArcGIS is the tool used by professionals (for various reasons), the information is generally available in VTK vector tiles as generated by ArcGIS from the the natural measure. Such vector tiles are used in viewer applications, like the online basemap viewers. Depending on the resolution of the envisaged TOAW map, a sensible choice for the requested zoom level of the tile set should be made. The resolution R of a zoom level z can be calculated using

$$R = \frac{\cos(LL)\ \pi\ r_{\mathrm{E}}}{2^{7+z}},$$

Table 1: Grid spacing *vs.* zoom levels

Grid spacing (km)	R (m/px)	z
200	2273	5 to 6
100	1136	6 to 7
50	568.2	7 to 8
25	284.1	8 to 9
20	227.3	8 to 9
15	170.5	9 to 10
10	113.6	9 to 10
5	56.82	10 to 11
2.5	28.41	11 to 12
1	11.36	13 to 14
0.5	5.682	14 to 15
0.25	2.841	15 to 16

with the radius of the earth r_E (6371.009 km, as given per the Wolfram Knowledgebase) and the requested latitude LL. For a latitude $LL = 47.5°$, *i. e.* central Europe, table 1 provides a guideline for the selection of the proper zoom level.

When a suitable tile set has been acquired, it needs to be converted into the non-proprietary mbfiles-format read by QGIS. A converter tool, specifically created due to the availability of the Austrian data, can be found at `https://github.com/BergWerkGIS/vtpk2mbtiles`.

The converted files should then be examined carefully, to learn and understand the data structure used. For each of the steps of this guide a merged, cleaned and repaired vector layer, containing the respective features, is required.

1.4 This package

What this package provides, is the necessary process description (this document), a set of style (qml) files, a folder of tileset textures applied by the style files, and two Python scripts. The style files and textures are used to render a TOAW-a-like map in QGIS; due to projection issues, it looks a bit off, but still provides a visualization of the creation progress. The first python file, `Coverage.py`, is for calculation of terrain coverage and provides a custom function to QGIS; the second, `Export.py`, is an export script to be run in QGIS that turns the constructed map layer into a TOAW-readable mml-file.

Part II
Process

The process is divided into several sections. Only section 2 is a mandatory step, everything following can be executed in any order.

Each subsubsection is named for a QGIS tool. The box contains all the necessary settings for that tool; anything that is not mentioned should not be changed. Occasionally further advice is provided.

The process will create a large number of temporary layers. It is recommended to delete them at the end of each section. Sometimes during a section multiple layers with the same name will be created. Those can either be renamed to tell them apart, or simply dragged to another position. QGIS supports drag and drop for the layer drop down fields, so the correct one (marked in the box) can be used.

The expressions given in the boxes can generally be simply copied. QGIS is in general not too picky regarding formatting. In some cases, generic parts of the expressions have to be replaced. This is noted below the box, but care has to be taken when layers have been renamed.

Generally, at the end of each section, a restart of QGIS is required. Sadly, a command to reload layer data has been suggested for years, but never implemented. Thus, a workaround is required.

If, at any step, QGIS complains about invalid meshes, applying the tool **Vector geometry — Fix geometries** should be sufficient. In that case, layer names in the process description may no longer apply.

The folder with the tileset textures has to be in the same folder where the package file is located as the style files use this path; this becomes important from 2.1.7 on.

2 Grid

Prerequisites:

- Nothing, in principle.

The first step is the creation of a hexagonal grid. As this grid is not perfect due to projection, unavoidable gaps have to be closed by self-snapping. The hex cells are then numbered using in-game logic. The grid will be formatted by application of a style file, as it will collect all the tile data over the course of this process.

From the created grid, connection lines (basically borders between the hexagons) have to be extracted and cleaned. These lines will be used to resolve path-type structures later on.

2.1 General grid

2.1.1 Vector creation — Create grid

Grid type:	Hexagon (Polgon)
Grid extent:	See below
Horizontal spacing:	$2/\sqrt{3}$ * *raster*
Vertical spacing:	*raster*

ATTENTION: The grid extent has to be sufficient to cover the whole map, plus a margin of at least a single tile. The top cell of the even rows will be discarded during export to match TOAW's grid structure.

The horizontal spacing is then calculated using the selected grid spacing (*raster*). The game uses a prefactor of 51/44 instead of $2/\sqrt{3}$. We will use the exact value, to minimize rounding errors.

2.1.2 Vector table — Drop field(s)

Input layer:	*Grid* (from 2.1.1)
Fields to drop:	`left`, `top`, `right`, `bottom`

2.1.3 Vector geometry — Snap geometries to layer

Input layer:	*Remaining fields* (from 2.1.2)
Reference layer:	*Remaining fields* (from 2.1.2)
Behavior:	Prefer aligning nodes, don't insert new vertices

2.1.4 Vector table — Field calculator

Input layer:	*Snapped geometry* (from 2.1.3)
Field name:	`X`
Result field type:	`Integer`
Expression:	`2*(x($geometry)-aggregate('snapped_geometry','min',x($geometry)))/ (sqrt(3)*raster)`

ATTENTION: Replace *snapped_geometry* by the actual one (from 2.1.3) from the Map Layers panel. Replace *raster* by the grid spacing.

2.1.5 Vector table — Field calculator

Input layer:	*Calculated* (from 2.1.4)
Field name:	`Y`
Result field type:	`Integer`
Expression:	`ceil(round((aggregate('calculated','max',y($geometry))-y($geometry))/ /raster-1,1))`

ATTENTION: Replace *calculated* by the actual one (from 2.1.4) from the Map Layers panel. Replace *raster* by the grid spacing.

2.1.6 Export layer

Export layer *Calculated* (from 2.1.5) as *Map* into a new package and rename it to *Map*.

2.1.7 Save the project

Save the project, if not done previously. The next step needs a valid project path.

2.1.8 Format layer

Format layer *Map* (from 2.1.7) by applying the corresponding QML-layer file to it. The number in the file name stands for the grid spacing in meters.

2.2 Connection lines

2.2.1 Vector geometry — Polygons to lines

Input layer:	*Map* (from 2.1.8)

2.2.2 Vector geometry — Explode lines

Input layer:	*Lines* (from 2.2.1)

2.2.3 Vector overlay — Intersection

Input layer:	*Exploded* (from 2.2.2)
Overlay layer:	*Exploded* (from 2.2.2)
Input fields to keep:	`id`
Overlay fields to keep:	`id`

2.2.4 Vector selection — Extract by expression

Input layer:	*Intersection* (from 2.2.3)
Expression:	`"id"<>"id_2"`

2.2.5 Export layer

Export layer *Matching features* (from 2.2.4) as *Connections* into the package and rename it to *Connections*.

3 Terrain

Prerequisites:

1. The grid from section 2.
2. A DEM (subsection 1.2) of the area to be mapped.

In this step, a slope analysis of the DEM will be performed. The result is then averaged over the grid cells. Terrain cutoff values have to be selected, that are used to build a terrain texture from the data. The texture is built by determining the terrain type of neighboring grid cells and calculating a contour sum. This sum is then appended to the grid.

Additionally, escarpments will be drawn as well, using the determined slope values as well as a cells average height.

3.1 Terrain analysis

3.1.1 Raster — Conversion — Translate

This command is not found in the toolbox, but in the toolbar. Compression is done as it takes only a few minutes, but cuts analysis time by around 50% — which can be hours in some cases.

Input layer:	Select the DEM (from subsection 1.2)
Advanced Parameters — Additional creation options — Profile:	High Compression

3.1.2 Raster terrain analysis — Slope

Elevation layer:	Select the DEM (from subsection 1.2)

3.1.3 Raster — Conversion — Translate

This command is not found in the toolbox, but in the toolbar. Compression is done as it takes only a few minutes, but cuts analysis time by around 50% — which can be hours in some cases.

Input layer:	*Slope* (from 3.1.2)
Advanced Parameters — Additional creation options — Profile:	High Compression

3.1.4 Raster analysis — Zonal statistics

Raster layer:	*Converted* (from 3.1.1)
Vector layer containing zones:	*Map* (from section 2)
Output column prefix:	`height_`
Statistics to calculate:	Mean

3.1.5 Raster analysis — Zonal statistics

Raster layer:	*Converted* (from 3.1.3)
Vector layer containing zones:	*Map* (from 3.1.4)
Output column prefix:	`slope_`
Statistics to calculate:	Median, Maximum

3.2 Manual inspection

Layer *Map* (from 3.1.5) needs to be manually analyzed to determine the terrain cutoff values. These are three values that determine the distinction between flat, hilly, mountainous and alpine terrain. Providing predetermined values is not a good choice, as this distinction depends upon the scale of the grid, the general area of the map, and the force composition.

Analysis can be performed by activating the Layer Styling Panel. Labels tied to the value of `slope_mean` should be activated. Additionally, graduated layer styling tied to the same value (with a suitable number of classes) should be activated (any existing stylings can be safely deleted). Therefore, each cell has a label and a color. Comparison with terrain features from an orthophoto can then be used to determine the cutoff values by — completely arbitrarily — assigning a terrain type to noteable terrain features, *i.e.* passes that have to be mountainous or alpine, ranges that have to be hilly, or similar. The graduated styling can then be adapted until there are only four colors left.

3.3 Generate terrain texture

3.3.1 Vector table — Field calculator

Input layer:	*Map* (from 3.1.5)
Field name:	`terrain`
Result field type:	Integer
Expression:	`CASE WHEN "slope_median" > alpine THEN 3 WHEN "slope_median" > mountain THEN 2 WHEN "slope_median" > hill THEN 1 ELSE 0 END`

ATTENTION: *alpine*, *mountain* and *hill* are the cutoff values that were determined in subsection 3.2.

3.3.2 Vector table — Field calculator

Input layer:	*Calculated* (from 3.3.1)
Field name:	`escarp`
Result field type:	Integer
Expression:	`CASE WHEN "slope_max" > 75 THEN 5 WHEN "slope_max" > 60 THEN 4 ELSE 0 END`

3.3.3 Vector general — Join attributes by nearest

Input layer:	*Calculated* (from 3.3.2)
Input layer 2:	*Calculated* (from 3.3.2)
Layer 2 fields to copy:	`id`, `height_mean`, `terrain`, `escarp`
Maximum nearest neighbors:	`6`
Maximum distance:	*raster*/2+1

ATTENTION: The maximum distance is calculated using the selected grid spacing. The added 1 is to account for rounding errors.

3.3.4 Vector table — Field calculator

Input layer:	*Joined layer* (from 3.3.3)
Field name:	`base2`
Result field type:	Integer
Expression:	`2^(3*azimuth(make_point(x($geometry),y($geometry)), make_point(x(geometry(get_feature_by_id('map_layer', attribute('id_2')))),y(geometry(get_feature_by_id('map_layer', attribute('id_2'))))))/pi())`

ATTENTION: Replace *map_layer* by the actual one (from section 2)) from the Map Layers panel.

3.3.5 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_06 FROM Calculated WHERE terrain > 0 AND terrain_2 > 0 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 3.3.4).

3.3.6 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_07 FROM Calculated WHERE terrain > 1 AND terrain_2 > 1 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 3.3.4).

3.3.7 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_08 FROM Calculated WHERE terrain > 2 AND terrain_2 > 2 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 3.3.4).

3.3.8 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_20 FROM Calculated WHERE escarp = 4 AND height_mean > height_mean_2 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 3.3.4).

3.3.9 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_21 FROM Calculated WHERE escarp = 5 AND height_mean > height_mean_2 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 3.3.4).

3.3.10 Vector general — Join attributes by field value

Input layer:	*Map* (from 3.1.5)
Table field:	`id`
Input layer 2:	*SQL Output* (from 3.3.5)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_06`

3.3.11 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 3.3.10)
Table field:	`id`
Input layer 2:	*SQL Output* (from 3.3.6)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_07`

3.3.12 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 3.3.11)
Table field:	`id`
Input layer 2:	*SQL Output* (from 3.3.7)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_08`

3.3.13 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 3.3.12)
Table field:	`id`
Input layer 2:	*SQL Output* (from 3.3.8)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_20`

3.3.14 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 3.3.13)
Table field:	`id`
Input layer 2:	*SQL Output* (from 3.3.9)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_21`

3.3.15 Vector table — Drop field(s)

Input layer:	*Joined layer* (from 3.3.14)
Fields to drop:	`height_mean`, `slope_median`, `slope_max`

3.3.16 Export layer

Export layer *Joined layer* (from 3.3.15) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map".

3.3.17 Save the project

Save the project and restart QGIS.

4 Coverage

Prerequisites:

1. The grid from section 2.
2. A (multi-)polygon layer containing all features counting as cropland of the area to be mapped.
3. A (multi-)polygon layer containing all features counting as forests of the area to be mapped.
4. A (multi-)polygon layer containing all features counting as settlements of the area to be mapped.
5. A (multi-)polygon layer containing all features counting as marshes of the area to be mapped.
6. A (multi-)polygon layer containing all features counting as fillers of the area to be mapped.

Initially, one has to understand which features from the base data are supposed to represent what type of in-game coverage. The features should be selected and grouped into the above four categories accordingly. To clarify here, if any category is missing, that type of coverage will (and can) simply not be generated. The respective steps in the process will then have to be skipped; in the calculations using the custom function, the respective entry needs to be replaced by a zero.

The filler is a special type of layer that incorporates features that should be considered in determining the coverage texture, but do not constitute a texture in themselves. The two most prominent types are the water layer (see section 5) and the traffic net (see section 6), that is a polygon feature (additionally to a (multi-)line layer) at higher zoom levels. Both of these types receive their own treatment. A filler is not strictly necessary, as the texture will generate without it, but will improve the quality of the final result.

The process computes a overlap percentage and uses a custom-written function to determine the coverage type for each cell. (It does so by calculating distances in five-dimensional parameter space to various defined anchor points.) Again, the contour sum is then calculated by determining the coverage type of neighboring cells to determine the coverage texture.

To use the custom function, the script file `Coverage.py` needs to be copied to `User\AppData\Roaming\QGIS\QGIS3\profiles\default\python\expressions` (for a Windows installation). In 4.1.2, the function needs to be activated by switching to the „Function Editor" tab and hitting „Save and Load Functions" to compile the Python script. It can only be used if it appears in the function list, under Coverage.

It is generally a good idea to fix the geometries before proceeding. Since feature size has a huge impact on computation time, it is almost mandatory to apply the following procedure beforehand on all the feature layers:

4.0.1 Vector geometry — Subdivide

Input layer:	Select the feature layer

Afterwards, the geometry of the layers is usually invalid and needs fixing. The fixed layers need to be renamed to *Cropland*, *Filler*, *Forest*, *Marsh*, and *Urban*, to make the following steps easier.

4.1 Coverage analysis

4.1.1 Vector analysis — Overlap analysis

Input layer:	*Map* (from 2)
Overlay layers:	*Cropland*, *Filler*, *Forest*, *Marsh*, *Urban*

4.1.2 Vector table — Field calculator

Input layer:	*Overlap* (from 4.1.1)
Field name:	`base2_09`
Result field type:	Integer
Expression:	`coverage_type('09',"Cropland_pc","Filler_pc","Forest_pc","Marsh_pc", "Urban_pc")`

4.1.3 Vector table — Field calculator

Input layer:	*Calculated* (from 4.1.2)
Field name:	`base2_10`
Result field type:	Integer
Expression:	`coverage_type('10',"Cropland_pc","Filler_pc","Forest_pc","Marsh_pc", "Urban_pc")`

4.1.4 Vector table — Field calculator

Input layer:	*Calculated* (from 4.1.3)
Field name:	`base2_13`
Result field type:	Integer
Expression:	`coverage_type('13',"Cropland_pc","Filler_pc","Forest_pc","Marsh_pc", "Urban_pc")`

4.1.5 Vector table — Field calculator

Input layer:	*Calculated* (from 4.1.4)
Field name:	`base2_14`
Result field type:	Integer
Expression:	`coverage_type('14',"Cropland_pc","Filler_pc","Forest_pc","Marsh_pc", "Urban_pc")`

4.1.6 Vector table — Field calculator

Input layer:	*Calculated* (from 4.1.5)
Field name:	`base2_15`
Result field type:	Integer
Expression:	`coverage_type('15',"Cropland_pc","Filler_pc","Forest_pc","Marsh_pc", "Urban_pc")`

4.1.7 Vector table — Field calculator

Input layer:	*Calculated* (from 4.1.6)
Field name:	`base2_16`
Result field type:	Integer
Expression:	`coverage_type('16',"Cropland_pc","Filler_pc","Forest_pc","Marsh_pc", "Urban_pc")`

4.1.8 Vector table — Field calculator

Input layer:	*Calculated* (from 4.1.7)
Field name:	`base2_28`
Result field type:	Integer
Expression:	`coverage_type('28',"Cropland_pc","Filler_pc","Forest_pc","Marsh_pc", "Urban_pc")`

4.1.9 Vector table — Field calculator

Input layer:	*Calculated* (from 4.1.8)
Field name:	`base2_29`
Result field type:	Integer
Expression:	`coverage_type('29',"Cropland_pc","Filler_pc","Forest_pc","Marsh_pc", "Urban_pc")`

4.2 Generate coverage texture

4.2.1 Vector general — Join attributes by nearest

Input layer:	*Calculated* (from 4.1.9)
Input layer 2:	*Calculated* (from 4.1.9)
Layer 2 fields to copy:	`id`, `base2_09`, `base2_10`, `base2_13`, `base2_14`, `base2_15`, `base2_16`, `base2_28`, `base2_29`
Maximum nearest neighbors:	`6`
Maximum distance:	*raster*/2+1

ATTENTION: The maximum distance is calculated using the selected grid spacing. The added 1 is to account for rounding errors.

4.2.2 Vector table — Field calculator

Input layer:	*Joined layer* (from 4.2.1)
Field name:	`base2`
Result field type:	Integer
Expression:	`2^(3*azimuth(make_point(x($geometry),y($geometry)), make_point(x(geometry(get_feature_by_id('map_layer', attribute('id_2')))),y(geometry(get_feature_by_id('map_layer', attribute('id_2'))))))/pi())`

ATTENTION: Replace *map_layer* by the actual one (from section 2)) from the Map Layers panel.

4.2.3 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_09 FROM Calculated WHERE base2_09 = 63 AND (base2_09_2 = 63 OR base2_10_2 = 63) GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 4.2.2).

4.2.4 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_10 FROM Calculated WHERE base2_10 = 63 AND (base2_10_2 = 63 OR base2_09_2 = 63) GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 4.2.2).

4.2.5 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_13 FROM Calculated WHERE base2_13 = 63 AND (base2_13_2 = 63 OR base2_14_2 = 63) GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 4.2.2).

4.2.6 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_14 FROM Calculated WHERE base2_14 = 63 AND (base2_13_2 = 63 OR base2_14_2 = 63) GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 4.2.2).

4.2.7 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_15 FROM Calculated WHERE base2_15 = 63 AND base2_15_2 = 63 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 4.2.2).

4.2.8 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_16 FROM Calculated WHERE base2_16 = 63 AND base2_16_2 = 63 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 4.2.2).

4.2.9 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_28 FROM Calculated WHERE base2_28 = 63 AND base2_28_2 = 63 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 4.2.2).

4.2.10 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_29 FROM Calculated WHERE base2_29 = 63 AND base2_29_2 = 63 GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 4.2.2).

4.2.11 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*SQL Output* (from 4.2.3)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_09`

4.2.12 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 4.2.11)
Table field:	`id`
Input layer 2:	*SQL Output* (from 4.2.4)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_10`

4.2.13 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 4.2.12)
Table field:	`id`
Input layer 2:	*SQL Output* (from 4.2.5)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_13`

4.2.14 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 4.2.13)
Table field:	`id`
Input layer 2:	*SQL Output* (from 4.2.6)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_14`

4.2.15 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 4.2.14)
Table field:	`id`
Input layer 2:	*SQL Output* (from 4.2.7)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_15`

4.2.16 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 4.2.15)
Table field:	`id`
Input layer 2:	*SQL Output* (from 4.2.8)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_16`

4.2.17 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 4.2.16)
Table field:	`id`
Input layer 2:	*SQL Output* (from 4.2.9)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_28`

4.2.18 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 4.2.17)
Table field:	`id`
Input layer 2:	*SQL Output* (from 4.2.10)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_29`

4.2.19 Export layer

Export layer *Joined layer* (from 4.2.18) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map”.

4.2.20 Save the project

Save the project and restart QGIS.

5 Water

Prerequisites:

1. The grid from section 2.
2. A (multi-)polygon layer containing all features of water of the area to be mapped.

Distributing water cells depending upon a simple overlap percentage is in principle possible, but will introduce discretization errors. This can be understood with the following example: A body of water can cover an area equal to a full hexagon, but be located between three adjacent cells. For all of those cells, the overlap percentage is below 50%, and hence no water cell will be drawn.

Furthermore, the contouring of the water cells does not happen in the cells themselves, but in the adjacent, non-water, cells, where the shore is located graphically.

Thus the calculated overlap values will be „fuzzified” by considering the overlap of the adjacent cells before the water cells are created. For contouring, the cells adjacent to water features a determined and receive an inverse contour.

The water layer need to be renamed to *Water* to make the following steps easier. Similarly to the coverage analysis, subdividing the water layer can improve computation speed.

5.1 Cumulative overlap analysis

5.1.1 Vector analysis — Overlap analysis

Input layer:	*Map* (from 2)
Overlay layers:	*Water*

5.1.2 Vector general — Join attributes by nearest

Input layer:	*Overlap* (from 5.1.1)
Input layer 2:	*Overlap* (from 5.1.1)
Layer 2 fields to copy:	`id`, `Water_pc`
Maximum nearest neighbors:	`6`
Maximum distance:	*raster*/2+1

ATTENTION: The maximum distance is calculated using the selected grid spacing. The added 1 is to account for rounding errors.

5.1.3 Vector general — Execute SQL

SQL Query:	`SELECT id, Water_pc*(1+SUM((1-ABS(Water_pc-Water_pc_2)/100)*Water_pc_2)/100) AS waterf_pc FROM 'Joined layer' WHERE Water_pc > 0 GROUP BY id, Water_pc`
Geometry type:	No geometry

ATTENTION: *Joined layer* (from 5.1.2).

5.2 Generate water texture

5.2.1 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 5.1.2)
Table field:	`id`
Input layer 2:	*SQL Output* (from 5.1.3)
Table field 2:	`id`
Layer 2 fields to copy:	`waterf_pc`

5.2.2 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 5.2.1)
Table field:	`id_2`
Input layer 2:	*SQL Output* (from 5.1.3)
Table field 2:	`id`
Layer 2 fields to copy:	`waterf_pc`

5.2.3 Vector selection — Extract by expression

Input layer:	*Joined layer* (from 5.2.2)

Expression:	`("waterf_pc" < 50 OR "waterf_pc" IS NULL)`
	`AND "waterf_pc_2" >= 50`

5.2.4 Vector table — Field calculator

Input layer:	*Matching features* (from 5.2.3)
Field name:	`base2`
Result field type:	Integer

Expression:	`2^(3*azimuth(make_point(x($geometry),y($geometry)),`
	`make_point(x(geometry(get_feature_by_id('map_layer',`
	`attribute('id_2')))),y(geometry(get_feature_by_id('map_layer',`
	`attribute('id_2'))))))/pi())`

ATTENTION: Replace *map_layer* by the actual one (from section 2)) from the Map Layers panel.

5.2.5 Vector general — Execute SQL

SQL Query:	`SELECT id, 63-base2_36 AS base2_36 FROM (SELECT id,`
	`CAST(ROUND(SUM(base2),0) AS INT) AS base2_36 FROM Calculated GROUP BY`
	`id) AS Hash WHERE base2_36 IS NOT NULL`

Geometry type:	No geometry

ATTENTION: *Calculated* (from 5.2.4).

5.2.6 Vector table — Field calculator

Input layer:	*Map* (from 2)
Field name:	`base2_11`
Result field type:	Integer

Expression:	`if(attribute(get_feature('sql_output','id',attribute('id')),`
	`'waterf_pc')>50,63,NULL)`

ATTENTION: Replace *sql_output* by the actual one (from 5.1.3)) from the Map Layers panel.

5.2.7 Vector general — Join attributes by field value

Input layer:	*Calculated* (from 5.2.6)
Table field:	`id`
Input layer 2:	*SQL Output* (from 5.2.5)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_36`

5.2.8 Export layer

Export layer *Joined layer* (from 5.2.7) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map”.

5.2.9 Save the project

Save the project and restart QGIS.

6 Paths

Prerequisites:

1. The grid from section 2.
2. A (multi-)line layer containing all rivers of the area to be mapped.
3. A (multi-)line layer containing all super rivers of the area to be mapped.
4. A (multi-)line layer containing both rivers and super rivers of the area to be mapped.
5. A (multi-)line layer containing all roads of the area to be mapped.
6. A (multi-)line layer containing all improved roads of the area to be mapped.
7. A (multi-)line layer containing both roads and improved roads of the area to be mapped.
8. A (multi-)line layer containing all rail lines of the area to be mapped.

Initially, one has to decide which features from the base data to include in which category; generally, there exists a much more detailed net of features than one would represent in even a map with the smallest grid size.

Creating paths is a bit more involved, as here not only the grid cells themselves are important, but also the connections between them. Paths can meander, and hence two adjacent cells containing the same path are not necessarily connected, as is the case for the texture.

The path intersections with both the cells and the connections lines have to be calculated. This information is then used to calculate the contour sum once again.

It is advised to make a single pass for the three type of paths. Building them all at once is possible, but will lead to a large number of equally-named temporary layers, and may lead to errors due to confusion.

If some types (say, roads) do not wish to be created, the appropriate steps can be skipped.

6.1 Rivers

6.1.1 Vector selection — Extract by location

Extract features from:	*Connections* (from 2)
By comparing to the features from:	Select the combined river layer

6.1.2 Vector selection — Extract by location

Extract features from:	*Map* (from 2)
By comparing to the features from:	Select the river layer

6.1.3 Vector selection — Extract by location

Extract features from:	*Map* (from 2)
By comparing to the features from:	Select the super river layer

6.1.4 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*Extracted (location)* (from 6.1.2)
Table field 2:	`id`
Layer 2 fields to copy:	`id`

6.1.5 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from6.1.4)
Table field:	`id`
Input layer 2:	*Extracted (location)* (from 6.1.3)
Table field 2:	`id`
Layer 2 fields to copy:	`id`

6.1.6 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 6.1.5)
Table field:	`id`
Input layer 2:	*Extracted (location)* (from 6.1.1)
Table field 2:	`id`
Layer 2 fields to copy:	`id_2`
Join type:	Create separate feature for each matching feature
Discard records which could not be joined:	✓

6.1.7 Vector table — Field calculator

Input layer:	*Joined layer* (from 6.1.6)
Field name:	`base2`
Result field type:	Integer
Expression:	`2^(3*azimuth(make_point(x($geometry),y($geometry)), make_point(x(geometry(get_feature_by_id('map_layer', attribute('id_2_2')))),y(geometry(get_feature_by_id('map_layer', attribute('id_2_2'))))))/pi())`

ATTENTION: Replace *map_layer* by the actual one (from section 2)) from the Map Layers panel.

6.1.8 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_23 FROM Calculated WHERE id_3 IS NULL GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 6.1.7).

6.1.9 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_24 FROM Calculated WHERE id_3 IS NOT NULL GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 6.1.7).

6.1.10 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*SQL Output* (from 6.1.8)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_23`

6.1.11 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 6.1.10)
Table field:	`id`
Input layer 2:	*SQL Output* (from 6.1.9)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_24`

6.1.12 Export layer

Export layer *Joined layer* (from 6.1.11) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map".

6.1.13 Save the project

Save the project and restart QGIS.

6.2 Roads

6.2.1 Vector selection — Extract by location

Extract features from:	*Connections* (from 2)
By comparing to the features from:	Select the combined road layer

6.2.2 Vector selection — Extract by location

Extract features from:	*Map* (from 2)
By comparing to the features from:	Select the road layer

6.2.3 Vector selection — Extract by location

Extract features from:	*Map* (from 2)
By comparing to the features from:	Select the improved road layer

6.2.4 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*Extracted (location)* (from 6.2.2)
Table field 2:	`id`
Layer 2 fields to copy:	`id`

6.2.5 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from6.2.4)
Table field:	`id`
Input layer 2:	*Extracted (location)* (from 6.2.3)
Table field 2:	`id`
Layer 2 fields to copy:	`id`

6.2.6 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 6.2.5)
Table field:	`id`
Input layer 2:	*Extracted (location)* (from 6.2.1)
Table field 2:	`id`
Layer 2 fields to copy:	`id_2`
Join type:	Create separate feature for each matching feature
Discard records which could not be joined:	✓

6.2.7 Vector table — Field calculator

Input layer:	*Joined layer* (from 6.2.6)
Field name:	`base2`
Result field type:	Integer
Expression:	`2^(3*azimuth(make_point(x($geometry),y($geometry)), make_point(x(geometry(get_feature_by_id('map_layer', attribute('id_2_2')))),y(geometry(get_feature_by_id('map_layer', attribute('id_2_2'))))))/pi())`

ATTENTION: Replace *map_layer* by the actual one (from section 2)) from the Map Layers panel.

6.2.8 Vector general — Execute SQL

SQL Query:
```
SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_32
FROM Calculated WHERE id_3 IS NULL GROUP BY id
```
Geometry type: No geometry

ATTENTION: *Calculated* (from 6.2.7).

6.2.9 Vector general — Execute SQL

SQL Query:
```
SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_33
FROM Calculated WHERE id_3 IS NOT NULL GROUP BY id
```
Geometry type: No geometry

ATTENTION: *Calculated* (from 6.2.7).

6.2.10 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*SQL Output* (from 6.2.8)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_32`

6.2.11 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 6.2.10)
Table field:	`id`
Input layer 2:	*SQL Output* (from 6.2.9)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_33`

6.2.12 Export layer

Export layer *Joined layer* (from 6.2.11) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map".

6.2.13 Save the project

Save the project and restart QGIS.

6.3 Rail

6.3.1 Vector selection — Extract by location

Extract features from:	*Connections* (from 2)
By comparing to the features from:	Select the rail layer

6.3.2 Vector selection — Extract by location

Extract features from:	*Map* (from 2)
By comparing to the features from:	Select the rail layer

6.3.3 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*Extracted (location)* (from 6.3.2)
Table field 2:	`id`
Layer 2 fields to copy:	`id`

6.3.4 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 6.3.3)
Table field:	`id`
Input layer 2:	*Extracted (location)* (from 6.3.1)
Table field 2:	`id`
Layer 2 fields to copy:	`id_2`
Join type:	Create separate feature for each matching feature
Discard records which could not be joined:	✓

6.3.5 Vector table — Field calculator

Input layer:	*Joined layer* (from 6.3.4)
Field name:	`base2`
Result field type:	Integer
Expression:	`2^(3*azimuth(make_point(x($geometry),y($geometry)), make_point(x(geometry(get_feature_by_id('map_layer', attribute('id_2_2')))),y(geometry(get_feature_by_id('map_layer', attribute('id_2_2'))))))/pi())`

ATTENTION: Replace *map_layer* by the actual one (from section 2)) from the Map Layers panel.

6.3.6 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_34 FROM Calculated GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 6.3.5).

6.3.7 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*SQL Output* (from 6.3.6)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_34`

6.3.8 Export layer

Export layer *Joined layer* (from 6.3.7) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map”.

6.3.9 Save the project

Save the project and restart QGIS.

7 Borders

Prerequisites:

1. The grid from section 2.
2. A (multi-)line layer containing all international borders of the area to be mapped.

Borders will be determined by transforming them into the area of the respective country. This process requires that the border line forms a closed loop; depending upon quality and resolution of the tileset, this is not guaranteed. Autosnapping (as in 2.1.3) can help, whereas large gaps in the data, but also due to the selection of the map area, have to be closed by hand.

The area polygon is then used to calculate a cell overlap, where the contouring is performed in a similar fashion as for water features. Again, subdividing the generated polygon layer will improve computation speed, but the created subdivision can sometimes wreak havoc to the layer format. In that case, **Vector geometry — Multipart to singleparts** can help.

7.1 Overlap analysis

7.1.1 Vector geometry — Polygonize

Input layer:	Select the border layer

7.1.2 Vector analysis — Overlap analysis

Input layer:	*Map* (from 2)
Overlay layers:	*Polygons* (from 7.1.1)

7.2 Overlap texture

7.2.1 Vector general — Join attributes by nearest

Input layer:	*Overlap* (from 7.1.2)
Input layer 2:	*Overlap* (from 7.1.2)
Layer 2 fields to copy:	`id`, `Polygons_pc`
Maximum nearest neighbors:	`6`
Maximum distance:	*raster*/2+1

ATTENTION: The maximum distance is calculated using the selected grid spacing. The added 1 is to account for rounding errors.

7.2.2 Vector selection — Extract by expression

Input layer:	*Joined layer* (from 7.2.1)
Expression:	`("Polygons_pc" >= '50' AND "Polygons_pc_2" < '50') OR ("Polygons_pc" < '50' AND "Polygons_pc_2" >= '50')`

7.2.3 Vector table — Field calculator

Input layer:	*Matching features* (from 7.2.2)
Field name:	`base2`
Result field type:	Integer
Expression:	`2^(3*azimuth(make_point(x($geometry),y($geometry)), make_point(x(geometry(get_feature_by_id('map_layer', attribute('id_2')))),y(geometry(get_feature_by_id('map_layer', attribute('id_2'))))))/pi())`

ATTENTION: Replace *map_layer* by the actual one (from section 2)) from the Map Layers panel.

7.2.4 Vector general — Execute SQL

SQL Query:	`SELECT id, CAST(ROUND(SUM(base2),0) AS INT) AS base2_44 FROM Calculated GROUP BY id`
Geometry type:	No geometry

ATTENTION: *Calculated* (from 7.2.3).

7.2.5 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*SQL Output* (from 7.2.4)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_44`

7.2.6 Export layer

Export layer *Joined layer* (from 7.2.5) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map”.

7.2.7 Save the project

Save the project and restart QGIS.

8 Annotations

Prerequisites:

1. The grid from section 2.
2. A (multi-)point layer containing all large font labels of the area to be mapped.
3. A (multi-)point layer containing all blue font labels of the area to be mapped.
4. A (multi-)point layer containing all default font labels of the area to be mapped.
5. A (multi-)point layer containing all small font labels of the area to be mapped.
6. A (multi-)point layer containing all peaks of the area to be mapped.
7. A (multi-)point layer containing all airfields of the area to be mapped.

By annotations we mean all features that do only exist in a single hex cell such that no contouring is necessary. This consists of the four types of standard labels as well as peaks and airfields.

In each case, the annotations will be joined with the cells based on location.

For the labels, it is expected that that the point layers contain the label text in a field named `label`.

8.1 Labels

8.1.1 Vector table — Field calculator

Input layer:	Select the layer with the labels for the first category
Field name:	`label`
Result field type:	String
Expression:	`'#1' + "label"`

8.1.2 Vector table — Field calculator

Input layer:	Select the layer with the labels for the second category
Field name:	`label`
Result field type:	String
Expression:	`'#2' + "label"`

8.1.3 Vector table — Field calculator

Input layer:	Select the layer with the labels for the third category
Field name:	`label`
Result field type:	String
Expression:	`'#3' + "label"`

8.1.4 Vector general — Join attributes by location

Base layer:	*Map* (from 2)
Join layer:	*Calculated* (from 8.1.1)
Fields to add:	`label`
Join type:	Take attributes of the first matching feature only
Discard records which could not be joined:	✓

8.1.5 Vector general — Join attributes by location

Base layer:	*Map* (from 2)
Join layer:	*Calculated* (from 8.1.2)
Fields to add:	`label`
Join type:	Take attributes of the first matching feature only
Discard records which could not be joined:	✓

8.1.6 Vector general — Join attributes by location

Base layer:	*Map* (from 2)
Join layer:	*Calculated* (from 8.1.3)
Fields to add:	`label`
Join type:	Take attributes of the first matching feature only
Discard records which could not be joined:	✓

8.1.7 Vector general — Join attributes by location

Base layer:	*Map* (from 2)
Join layer:	Select the layer with the labels without a category
Fields to add:	`label`
Join type:	Take attributes of the first matching feature only
Discard records which could not be joined:	✓

8.1.8 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*Joined layer* (from 8.1.4)
Table field 2:	`id`
Layer 2 fields to copy:	`label`

8.1.9 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 8.1.8)
Table field:	`id`
Input layer 2:	*Joined layer* (from 8.1.5)
Table field 2:	`id`
Layer 2 fields to copy:	`label`

8.1.10 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 8.1.9)
Table field:	`id`
Input layer 2:	*Joined layer* (from 8.1.7)
Table field 2:	`id`
Layer 2 fields to copy:	`label`

8.1.11 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 8.1.10)
Table field:	`id`
Input layer 2:	*Joined layer* (from 8.1.6)
Table field 2:	`id`
Layer 2 fields to copy:	`label`

8.1.12 Vector table — Field calculator

Input layer:	*Joined layer* (from 8.1.11)
Field name:	`label`
Result field type:	String
Expression:	`CASE WHEN "label" IS NOT NULL THEN "label" WHEN "label_2" IS NOT NULL THEN "label_2" WHEN "label_3" IS NOT NULL THEN "label_3" WHEN "label_4" IS NOT NULL THEN "label_4" END`

8.1.13 Vector table — Drop field(s)

Input layer:	*Calculated* (from 8.1.12)
Fields to drop:	`label_2`, `label_3`, `label_4`

8.1.14 Export layer

Export layer *Remaining fields* (from 8.1.13) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map".

8.1.15 Save the project

Save the project and restart QGIS.

8.2 Symbols

8.2.1 Vector selection — Extract by location

Extract features from:	*Map* (from 2)
By comparing to the features from:	Select the airfield layer

8.2.2 Vector table — Field calculator

Input layer:	*Extracted (location)* (from 8.2.1)
Field name:	`base2_39`
Result field type:	Integer
Expression:	`2`

8.2.3 Vector general — Join attributes by field value

Input layer:	*Map* (from 2)
Table field:	`id`
Input layer 2:	*Calculated* (from 8.2.2)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_39`

8.2.4 Vector selection — Extract by location

Extract features from:	*Map* (from 2)
By comparing to the features from:	Select the peak layer

8.2.5 Vector table — Field calculator

Input layer:	*Extracted (location)* (from 8.2.4)
Field name:	`base2_39`
Result field type:	Integer
Expression:	`4`

8.2.6 Vector general — Join attributes by field value

Input layer:	*Joined layer* (from 8.2.3)
Table field:	`id`
Input layer 2:	*Calculated* (from 8.2.5)
Table field 2:	`id`
Layer 2 fields to copy:	`base2_39`

8.2.7 Vector table — Field calculator

Input layer:	*Joined layer* (from 8.2.6)
Field name:	`base2_39`
Result field type:	Integer
Expression:	`CASE WHEN "base2_39" IS NOT NULL THEN "base2_39" WHEN "base2_39_2" IS NOT NULL THEN "base2_39_2" END`

8.2.8 Vector table — Drop field(s)

Input layer:	*Calculated* (from 8.2.7)
Fields to drop:	`base2_39_2`

8.2.9 Export layer

Export layer *Remaining fields* (from 8.2.8) as *Map* into the package (overwriting the old layer) — Uncheck „Add saved file to map".

8.2.10 Save the project

Save the project and restart QGIS.

9 Exporting

9.1 Plugins — Python Console — Show Editor — Open Script...

Open the script file `Export.py` and run it. It creates a TOAW readable `Export.mml` map in the same directory as the QGIS project. The export can be performed at any time; it will simply skip fields that it cannot find in the *Map* layer.