QHG Manual

Christoph Zollikofer's group

Department of Anthropology
University of Zurich
Zurich, Switzerland

4th July 2018

Contents

1 About QHG

The main concept of QHG3 are populations (base class SPopulation). A population consists of an array of agent structs and a collection of actions (base class Action) "glued" together by code. The main application manages a set of populations, letting them perform their actions in every iteration of the simulation. It also manages input data and handles timed events (e.g. climate change), and creates output files of environmental or population data at freely specifiable times. For an analysis of colonization of Americas using QHG, see reference [1].

Figure 1 illustrates the coarse structure of the QHG simulation program.

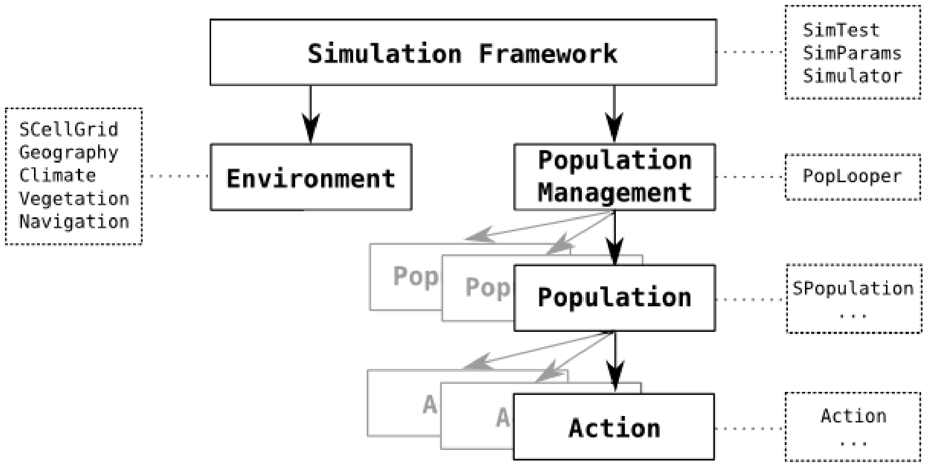

Figure 1: **The structure of QHG.** The arrows between the components (solid border) denote ownership. The boxes with dashed borders list the main classes associated with a component, with '...' denoting derived classes. **SimTest** opens the log file, creates a Simulator object and runs it. **SimParams** base class for Simulator with methods for reading program options, creating objects and loading data. **Simulator** runs the simulation loop and handles events. It is parallelized with openMP. **PopLooper** manages the SPopulation objects and issues commands from Simulator to all of them. **SPopulation** base class for all population classes; only derived classes can be used as actual populations. **Action** virtual base class for all action classes; only derived classes can be used as actual actions. **SCellGrid** the grid on which the simulation is running: agents sit on nodes and can travel to connected nodes. **Geography** contains longitudes, latitudes and altitudes of the nodes, as well as the water-related data. **Climate** contains temperatures and precipitation for each cell. **Vegetation** contains net primary production for each cell. **Navigation** contains over seas connections for some nodes. You can find the code of each module in a file named after the module.

2 Installation

To compile QHG, use the following commands:

```
$cd /path/to/QHG3
$OPT=1 make clean QGHMain
#(substitute "/path/to/QHG3" with your real path to the QHG3 main
    directory)
# or use
$make clean QHGMain
# see if it works:
$app/QHGMain
```

Furthermore, you need a output directory for your simulations (every simulation creates a subdirectory in this output directory)

```
mkdir ~/Simulations
```

Of course you can create it anywhere you want and call it whatever you want.

If you get an error that cgcc or gcc is not found, use the following command:

```
$export GCC=g++
```

2.0.1 Installation on Ubuntu 16.04

1. Install necessary packages

   ```
           $sudo apt update
           $sudo apt install g++ make gdb valgrind
           $sudo apt install zlib1g-dev libssl-dev libgsl0-dev
           $sudo apt install libhdf5-serial-dev
   ```

2. Add environment variables to /.profile For the compilation to work we must make sure some environment variables will always be set. For this you have to edit the file '.profile' in your home directory.

   ```
           vim ~/.profile
   ```

   Add the following lines to the end of ' /.profile'

   ```
       export CPATH=/usr/include/hdf5/serial/:$CPATH
       export
           LIBRARY_PATH=/usr/lib/x86_64-linux-gnu/hdf5/serial/:$LIBRARY_PATH
       export GCC=/usr/bin/g++
   ```

   Save and close the file

3. Create the source code directory and download the code.
4. Close and open the terminal.
5. Install packages needed by QHG tools.

```
        sudo apt install libpng-dev
        sudo apt install libglib2.0-dev
        sudo apt install libcairo2-dev
        sudo apt install libgtk2.0-dev
```

6. Modify /.profile again.

   Add the following line (no line break) to the end of the file:

```
     export CPATH=/usr/include/cairo/:/usr/include/gtk-2.0/gdk/:
     /usr/lib/x86_64linux-gnu/glib-2.0/include/:
     /usr/include/glib-2.0:$CPATH
```

7. Close and open the terminal
8. Build the tools

```
        $cd ~/progs/QHG3
        $make tools_n
```

3 Main parameters

You can run the program without any parameters to check a list of available parameters:

```
$./QHGMain

usage
-h,                        show help
--help=<topic>             show help for topic (use name of option or
    "all")
--log-file=<filename>      set name of log file (default:"SimTest.log")
--grid=<grid-file>         set grid file
--num-iters=<iters>        set number of iterations
--geo=<geo-file>           set geography file
--climate=<climate-file> set climate file
--veg=<veg-file>           set vegetation file
--nav=<nav-file>           set navigation file
--pops=<pop-list>          set population files
--output-prefix=<name>     prefix for output files (default: "output_")
--output-dir=<dirname>     output directory (default: "./")
--data-dirs=<dirnames>     data directories (default: "./")
--read-config=<conf>       read config from file <conf>
--write-config=<conf>      write config to file <conf>
--events=<event-list>      set events
```

```
  -n <iters>              set number of iterations
  --layer-size=<size>      set layer size (default: 65536)
  --shuffle=<num>          shift random generator's sequence (default: 0)
  --seed=<seedtype>        set seed for random number generators
  --pop-params=<paramstring> set special population parameters
  --zip-output             use gzip to zip all output qdf files
```

I will go through each of these parameters one by one and explain what they do. Next, I will explain their values in detail.

3.1 $--log-file=<filename>$ - Optional

If you provide a file path, QHG will save the essential information during execution. It will be useful for debugging.

3.2 $--grid=<grid-file>$ - Required

Here, you should provide a grid file in QDF format (see section 4.1). A grid file will have information about the shape and size of the universe and number of cells in the universe. It can additionally contain geography, climate, and vegetation data.

3.3 $--num-iters=<iters>$ - Required

Number of years the model will run from the past up to the present.

3.4 $--geo=<geo-file>$ - Optional

If information about geography are not provided in the grid file (section 4.2), it can be provided separately here as a QDF file. It must contain data for the same number of cells as the grid file.

3.5 $--climate=<climate-file>$ - Optional

If information about climate are not provided in the grid file (section 4.3), it can be provided separately here as a QDF file. It must contain data for the same number of cells as the grid file.

3.6 $--veg=<veg-file>$ - Optional

If information about vegetation are not provided in the grid file (section 4.4), it can be provided separately here as a QDF file. This will not be needed when you have NPP data, because vegetation is used to calculate NPP. It must contain data for the same number of cells as the grid file.

3.7 $--nav=<nav-file>$ - Optional

Provide a navigation file, which includes information about corridors between islands, for example.

3.8 $--pops=<pop-list>$ - Required

pop-list is a comma-separated list of QDF-files, each containing population data. It must contain data for the same number of cells as the grid file. The population files are result of previous run of simulations. Alternatively, if you want to start from scratch, you may use $--pops=<cls-file>:<data-file>$, where cls-file is a class definition file, and data-file is a file containing agent data (see section 4.5).

3.9 $--output-prefix=<name>$ - Optional

A prefix that is prepended to all output files. This will be useful when you run many simulations with different parameters. There are two kind of output files, both of which are in QDF format. One output file stores population specific details at certain time points specified by the events parameter (section 3.14), and the other output file stores information about geography, vegetation, climate etc.

3.10 $--output-dir=<dirname>$ - Optional

A directory in which to save the result files.

3.11 $--data-dirs=<dirnames>$ - Optional

If you provide this option, you will not have to write the full path to each data file, rather the program will look for those files in the directories you provide here as a comma-separated list.

3.12 $--write-config=<conf>$ - Optional

If you provide a file name, QHG will save the configurations of the current simulation in the file. You may later import the config file in a new simulation and use the exact same parameters. If you import the config file and you provide new parameter values on the command line, the new values will overwrite the values in the config file.

3.13 $--read-config=<conf>$ - Optional

Import parameter values from a previously run simulation here.

3.14 $- - events =< event - list >>$ - Optional

This parameter specifies certain events that happen during a simulation; events such as writing data at certain time points, changing the environment, and introducing new populations.

```
The event-list string is a comma-separated list of events:
   event-list      ::= "'" <event> ["," <event>]*"'"
The entire event list must be enclosed in quotes;
if the event list contains environment variables, use double quotes.
   event           ::= <event-type> "|" <event-params> "@"
        <event-times>
   event-type      ::= "write" | "env" | "arr" | "pop" | "file" |
        "check" | "comm"
   <event-times> defines at which times the event should be triggered.
   event-times     ::= <full-trigger> [ "+" <full-trigger>]*
   full-trigger    ::= <normal-trigger> | <point-trigger>
   normal-trigger  ::= [<trigger-interval>] <step-size>
   trigger-interval ::= "["[<minval>] ":" [<maxval>] "]"
This form causes events to be triggered at time
     <minval>+k*<step-size>, for
k = 0 ... (<maxval>-<minval>)/<step-size>.
If <minval> is omitted, it is set to fNegInf,
if <maxval> is omitted, it is set to fPosInf.
If <trigger-interval> is omitted, it is set to [fNegInf:fPosInf].
   point-trigger   ::= "["<time>"]"
This form causes a single event at the specified time.
Example:
   20+[305:]500+[250:600]10+[37]
   fire events every 20 steps,
   and every 500 steps starting from step 305,
   and every 10 steps between steps 250 and 600,
   and a single event at step 37.

The event parameters differ for the event types:
for <event-type> == "write":
event-params ::= <type> ["+" <type>]*
type         ::= "grid" | "geo" | "climate " |
              "veg" | "stats" | "nav" | "pop:"<speciesname>["#"]
Write the specified data sets to file.
In case of "pop", appending a '#' to the populaton name prevents
     writing of additional data (e.g. genome).

for <event-type> == "env":
event-params  ::= <type> ["+" <type>]* ":" <qdf-file>
type          ::= "geo" | "climate" | "veg" | "nav"
Read the specified data sets from file.

for <event-type> == "arr":
event-params  ::= <type> ":" <arrayname> ":" <qdf-file>
type          ::= "geo" | "climate" | "veg"
Read the specified arrays from file.

for <event-type> == "pop":
```

```
event-params  ::= <speciesname> ["+" <speciesname>]* ":" <qdf-file>
Read the specified populations from file.
for <event-type> == "pop":
event-params  ::= <speciesname> ["+" <speciesname>]* ":" <qdf-file>
Read the specified populations from file.

for <event-type> == "file":
event-params  ::= <filename>
The file must contain lines of the form
line          ::= <event-type> "|" <event-params> "@" <event-times>
Add the events listed in the file to the event manager.

for <event-type> == "comm":
   event-params  ::= <cmd-file> | <command>
   command       ::= <iter_cmd> | <del_action_cmd> | <event>
   iter_cmd      ::= "SET ITERS" <num_iters>
   del_action_cmd ::= "REMOVE ACTION" <population>:<action_name>
   event         : any event description; see definition above
The format of <cmd-file>
   cmd-file      ::= <command-line>*
   command-line  ::= <command> <NL>
If the <cmd-file> has changed since the last triggered time the
    contents will be executed:
   iter_cmd:      set new iteration limit
   del_action_cmd: remove action from prio list
   event:         add event to event manager's list

for <event-type> == "check":
event-params  ::= <what> ["+" <what]*
what          ::= "lists"
(Debugging only) "lists": check linked lists at specified times

Full example:
--events='env|geo+climate:map120.qdf@[120],write|pop:GrassLover|grid@5+[20:30]1,comm:REMOVE
    ACTION ConfinedMove@[3000]'
This loads a QDF file containing new geography, vegetation and
    climate data
at step 120, and writes a QDF file containing grid data
and population data for species "GrassLover" every 5 steps
and additiionally every step between steps 20 and 30.
```

3.15 $--layer-size=<size>$ - Optional

Size of an array to store individuals as classes. If you use too small a size, it will result in creating too many new arrays. If too large, it will occupy too much memory. Default is 65536.

3.16 $--shuffle=<num>$ - Optional

This is done by generating ¡num¿ random numbers on each random number generator before the start of the simulation, i.e., this option can be used to change the random number sequence. If other parameters are the same, two simulations with the same shuffle number will have exactly the same output.

3.17 $--seed=<seedtype>$ - Optional

You can choose the seeds to start the simulation with. Choosing the same seed across different simulations with the same parameter values and number of processors will result in the same output.

```
--seed=<seedtype>  (technical) seed the random generators
seedtype  ::= "random" | "seq:"<sequence> | "file:"<seed-file> |
    "phrase:"<arbitrary>
sequence  ::= (<hexnumber> ",")*
arbitrary : arbitrary string (enclose in quotes if it contains spaces)
seed-file : name of a seed file
If seedType is "random", the seeds are filled with random numbers
based on the current time
A sequence must be consist of 16 comma-separated 8-digit hexnumbers.
If sequence is given, it is passed directly to the random number
    generators.
If an arbitrary string is provided, it is turned into a seed sequence
    using
a hash function (SHA-512)
If a seed file is specified it must conform to the following format.
The general format of a seed file:
seed-file ::= <seed-def>*
seed-def ::= <header> <seed-line>* <footer>
header   ::= "SEED"[:<dest>]
seed-line ::= (<hexnumber> ",")*
dest     :   A string describing which RNG the sequence should be
        passed to (NOT IMPLEMENTED YET)
footer   ::= "SEED_END"
In total, 16 8-digit hex numbers must be given between header and
    footer
Lines starting with "#" are ignored
```

3.18 $--pop-params=<paramstring>$ - Optional

This currently does not do anything.

3.19 $--zip-outout$ - Optional

To save disk space, this will zip output files immediately after generation.

4 Parameter values

4.1 Grid file

Grid files, as well as many other files, contain data in the QDF format (section 7.2). A grid file is a dataset with three columns and as many rows as there are cells on the grid. First column is cell ID (CellID), second is number of neighbors of the cell (NumNeighbors), and the last column is the ids of the neighbors as a list (Neighbors).

Examples can be found in the resources/grids folder in the main QHG folder. In the file names, S means the file contains grid data, G denotes the grid file also contains geography, likewise, C stands for climate, V for vegetation and N for navigation data. ”ieq” in the file names means the grid is an equal are icosahedron subdivision. If the file name contains an expression of the form $< width > \times < height >$ the grid is rectangular with the indicated size.

New grids can be produced with the tool `EQTileTool` in the `tools_ico` folder (see section 7.1).

4.2 Geo file

It is a QDF file that contains different datasets, each comprising of a list with information about area, altitude, longitude, latitude, water, and ice coverage of each cell in the grid, and distance between each cell to its neighbors.

4.3 Climate file

It is a QDF file that contains values for rain falls and temperatures for each cell in the grid. These information can also be an object in the grid file.

4.4 Vegetation file

A vegetation file is a QDF file containing a data set of two columns, ”BaseNPP” and ”NPP”. The ”NPP” column is usually calculated from ”BaseNPP” and NPP increases due to rivers or coastal contributions.

4.5 Population files

To start a population for the first time, you will need two files, one for the definition of a population, a cls file, and one containing each individual’s location, birth year, and sex. You can find examples of these files in the `resources/pops` folder.

4.5.1 Population definition file

Below is an example definition file, which I will use to explain different parameters:

```
CLASS OoANavGenPop
SPECIES 119 Sapiens_ooa
SENSING 1

BEGIN_PARAMS

#-- for WeightedMove
WeightedMoveProb = 0.2
# This is the probability of the movement of agents

#-- for Navigate
NavigateDecay   = -0.001
NavigateDist0   = 150.0
NavigateProb0   = 0.1
NavigateMinDens = 0
# NavigateDecay uses an exponential function to calculate the
    probability that an individual will pass through corridors
    that connects Australia and islands in South East Asia to the
    mainland Asia. NavigateDist0 takes a value of number of
    kilometers (here 150). NavigateProb0 is the probability that
    an agent will get to this distance. Given these information,
    the code will calculate the probability of reaching any other
    distance based on a logistic decay function.

#-- for SingleEvaluator Alt
AltCapPref = -0.1 0 0.1 0.01 1500 1.0 2000 1 3000 -9999
# This parameter defines the preference of different altitude for
    the individuals. Values are read in pairs. The first value
    defines the altitude and the second values defines how much
    that altitude is preferred by the agents. Any altitudes
    between two points in this vector have a preference which is a
    linear function of the preference of these two points. One can
    provide any value for the preferences. If you come from 1 to
    0, you will have a less steep line than when you come from 1
    to -1000.

#-- for SingleEvaluator NPP
NPPPref = 1 0 32 1 1000 1 10000 1
# Similar to AltCapPref, this parameter defines the preference of
    different NPP (net primary production) values by the agents.
    Values are read in pairs, where the first value in each pair
    is the NPP of the environment, and the second one is how much
    this NPP is preferred by the agents. One can provide any value
    for the preferences. If you come from 1 to 0, you will have a
    less steep line than when you come from 1 to -1000.

#-- for MultiEvaluator Alt+NPP
AltWeight = 0.2
NPPWeight = 0.8
```

```
# AltWeight and NPP weight define how much altitude and NPP are
    important in the decision of the agents.


#-- for VerhulstVarK
Verhulst_b0 = 0.2
Verhulst_d0 = 0.001
Verhulst_theta = 0.01
# These are important parameters. Verhulst_b0 is the initial birth
    rate of the population, Verhulst_d is the initial death rate,
    and Verhulst_theta is the turnover of the population. Check
    Figure 1 in ref \cite{Tkachenko2017} for how they are related.


#-- for NPPCapacity
NPPCap_K_min              = 6.0
NPPCap_NPP_max            = 2600.0
NPPCap_NPP_min            = 160.0
NPPCap_coastal_factor     = 0.2
NPPCap_coastal_max_latitude = 66.0
NPPCap_coastal_min_latitude = 50.0
NPPCap_max_capacity       = 60.0
NPPCap_water_factor       = 0.5
#NPPCap_coastal_factor adds to actual NPP (net primary production)
    of coastal lands in latitudes between
    NPPCap_coastal_min_latitude and NPPCap_coastal_max_latitude .
# NPPCap_water_factor: increases the carrying capacity (cc) in a
    cell with water using cc' = cc*(1+NPPCap_water_factor)

#To explain some of the above variables, I'll rename them for
    easier reading
# Kmin: NPPCap_K_min
# Kmax: NPPCap_max_capacity
# NPPmin: NPPCap_NPP_min
# NPPmax: NPPCap_NPP_max
# To convert NPP values to Carrying capacities i use a ramp
    function:
# if npp < NPPmin
# cc = Kmin
# else if npp < NPPmax
# cc = Kmin + (npp -NPPmin)*(KMax - Kmin)/(NPPmax - NPPmin)
# else
# cc = Kmax


#-- for Fertility
Fertility_interbirth     = 2.0
Fertility_min_age        = 15.0
Fertility_max_age        = 50.0
# Fertility_interbirth is the min years between two children.
    Fertility_min_age and Fertility_max_age only affect females.
    They cannot have children out of that range.
```

```
#-- for Genetics
Genetics_bits_per_nuc    = 1
Genetics_create_new_genome = 1
Genetics_genome_size     = 8192
Genetics_initial_muts    = variants:3
Genetics_mutation_rate   =  2.00000000e-06
Genetics_num_crossover   = -1
# values related to how genetic data is stored.
    Genetics_bits_per_nuc specifies whether genomes are binary
    (equal to a value of 1), or they can store four different
    bases (equal to a value of 2). If genomes are binary, an
    infinite sites model is used, so that no site will experience
    back mutations. Genetics_create_new_genome should be 1 only if
    you are creating a new population. If you use a population
    from a previous run of simulation, set this to 0, or the whole
    information in the genome will be lost. Genetics_genome_size
    is the number of segregating sites in the genome which have
    full linkage disequilibrium. Genetics_initial_muts says how
    many variants should exist a population at time 0.
    Genetics_mutation_rate is the mutation rate per nucleotide per
    genome per generation. Genetics_num_crossover is the number of
    crossingovers that occur among the haplotypes of each parent
    during meiosis. A value of -1 indicates free recombination.
    Note that the current implementation of crossing over does not
    include a recombination rate. Thus if you crossing over
    happens, it happens in EVERY meiosis.

#-- for old age death
OAD_max_age              = 60.0
OAD_uncertainty          = 0.1
# OAD_max_age is the maximum age an agent will be alive with an
    uncertainty of OAD_uncertainty.


#-- priorities
PRIO NPPCapacity      1
PRIO Fertility        2
PRIO RandomPair       3
PRIO MultiEvaluator   5
PRIO VerhulstVarK     6
PRIO WeightedMove     7
PRIO Navigate         8
PRIO GetOld           9
PRIO OldAgeDeath      10
# The priorities define which one the functions run before others
    at each time step.

END_PARAMS
```

4.5.2 Population data file

The population data file has the following format:

```
# lon lat: life id birth gender mass age
34.9155 4.93459 : 1 1 -10 0 0 0
34.9155 4.93459 : 1 2 -10 1 0 0
34.9155 4.93459 : 1 3 -10 0 0 0
34.9155 4.93459 : 1 4 -10 1 0 0
34.9155 4.93459 : 1 5 -10 0 0 0
34.9155 4.93459 : 1 6 -10 1 0 0
34.9155 4.93459 : 1 7 -10 0 0 0
34.9155 4.93459 : 1 8 -10 1 0 0
```

Where each line represents an individual at location log and lat. life 1 means the individual is alive and 0 means its dead. gender 0 is female and 1 is male. Birth is date of birth. A negative value denotes a birth time prior to the simulation start (this way we can start a simulation with adult individuals). mass and age should be left at zero as they are deprecated.

5 Output of QHG

Basically, all output of QHG is in the form of QDF files. Usually all data regarding the environment (grid, geography, etc.) is bundled into an output QDF file, whereas population data is stored in a separate QDF file. What data should be out at what time can be specified with the `--event parameter` (see section 3.14).

6 Examples

Here is how you call QHGMain (you must write the contents of the entire box onto a single line; make sure you separate the options with a space):

```
$/path/to/QHG3/app/QHGMain
    --data-dirs='../data/gridpreparation,../data/pops'
--events='write|pop:Sapiens_ooa@20000,write|grid+geo+climate+veg+nav+stats@20000'
--grid=GridSGCw10_20k_ieq_256.qdf --log-file=test_1.log
    --output-dir=~/Simulations/test1 --output-prefix=ooa
--pops='clsOoANavGen.def:SapiensData_K60.txt' --shuffle=171829 -n
    80000 > test1.out
```

This will run a simulation with the agents described in clsOoANavGen.def for 80000 steps and write the data to the directory ~/Simulations/test_1. Every 20000 steps it will write a snap shot of the population and of the environment. During the simulation, the events in the file navworld_080k.evt are executed: every 1000 steps (i.e. years) a new environment is loaded. The logfile `test_1.log` is written to the current directory. The output of QHG is very detailed and contains a lot of technical and debug information, as well as some statistics. Therefore, it is advisable to redirect stdout into a file, as shown in the example. After the simulation has successfully completed, this file may be deleted.

7 Other tools

7.1 EQTileTool

This tool is for creating new grid files.

```
./EQTileTool - tiling for EQsahedron
usage:
./EQTileTool "ico" -n <subdivnodes> -t <subdivtiles> [-q
    <qdfbody>] [-i <ignbody>] [-v <verbosity>]
(creates a tiling of the grid based on a coarse subdivision of the
    EQsahedron)
or
./EQTileTool "rect" -n <subdivnodes> -t <nLon>"x"<nLat>":"<maxLat>
    [-q <qdfbody>] [-i <ignbody>] [-v <verbosity>]
(creates Lon-Lat rectangular tiles plus two caps at the poles)
where
subdivnodes subdivision of nodes
subdivtiles subdivision of tiles (use 0 for an untiled grid)
nLon        number of tiles in longitudinal direction
nLat        number of tiles in latudinal direction
maxLat      maximum latitude (N and S) for tiles; caps start above
    this latitude
qdfbody     body for QDF output (no qdf output if omitted
ignbody     body for IGN output (no ign output if omitted
verbosity   verbosity level - 0 : NONE, 1 : TOPLEVEL, 2 : INFO, 3
    : DETAIL (default: "INFO")

NOTE: subdivnodes > 0, subdivtiles >= 0
NOTE: (subdivnodes + 1) must be divisible by (subdivtiles + 1)
   i.e. tile grid nodes must be a subset of the grid nodes
NOTE: (subdivnodes + 1)/(subdivtiles + 1) must be greater than 2
   i.e. avoid "faceless" tiles
```

7.2 QDF file format

To view QDF files in the bash environment, you may use the h5dump program:

```
# -A flag shows only the headers and info about the data, but not
    the whole tables.
$h5dump -A yourFile.qdf
```

You may also use the VisIt program (see section 7.3).

7.3 VisIt

Using VisIt program, you can map simulation info, such as population movements and demography, or geography and climate, into the simulation universe (grid) and view the data. You need to install a special plugin developed by the group to view QDF files in VisIt. To map population to a grid and see them in VisIt, use `LinkQDFGroups.py` program (section 7.4).

VisIt is a visualization created and maintained by the Lawrence Livermore NAtional Library. Using the visit-plugin for the QDF format, most of the data of all types of QDF files can be displayed as pseudocolor maps on the grid. To display data from a pure population QDF file you must link it to a grid with the `LinkQDFGroups.py` program (section 7.4).

In order to build and use this plugin, VisIt sources should be downloaded from `https://wci.llnl.gov/simulation/computer-codes/visit/source` and be built with the `build_visit` script found at the same site. After successful installation, look at the file `BUILD_PLUGIN.sh` and set environment variables as indicated in this file.

7.4 LinkQDFGroups

This program is for mapping a grid file to population file, so that you can see populations in your universe in the VisIt program.

```
$LinkQDFGroups.py <population QDF file> <grid file> SGCVNM
# SGCVNM defines the data to take from the grid file, where S
    stands for grid, G for geography, C for climate, V for
    vegetation, N for navigation, and M for move statistics.
```

7.5 pop_attrs.py

This tool allows you to view and modify attributes of a QDF population file.

Usage:

```
./pop_attrs.py show [all] <pop-qdf>[:<pop-name>] [<attr-name>]*
     or:
        ./pop_attrs.py mod <pop-qdf>[:<pop-name>] (<attr-name>
            <attr-val>)*
     or:
        ./pop_attrs.py add <pop-qdf>[:<pop-name>] <attr-name>
            <attr-val> [<attr_type>]
     or:
        ./pop_attrs.py del <pop-qdf>[:<pop-name>] <attr-name>

     where:
        pop-qdf   QDF file with populations
        pop-name  name of population (if omitted, the first
            population is used)
        attr-name name of attribute
        attr-val  new value for attribute (the value must have the
            correct type
        attr_type a numpy type such as 'float64', 'int32', 'S2' etv
```

Note that the ”mod” variant currently only allows changing the values of numerical attributes.

7.6 longlatforid.py

With this tool you can get the longitude, latitude, altitude and ice cover for node-IDs of a particular grid.

Usage:

```
    ../useful_stuff/longlatforid.py <qdf-file> <output-mode> <nodeid>*
    where
       qdf-file    a qdf file with Grid and Geography grtoup
       output-mode "csv" or "nice"
       nodeid      node id for which to extract data
    Example
       ../useful_stuff/longlatforid.py navworld_085_kya_256.qdf 123
           4565 7878 31112 0
```

References

[1] Natalie Tkachenko, John D. Weissmann, Wesley P. Petersen, George Lake, Christoph P. E. Zollikofer, and Simone Callegari. Individual-based modelling of population growth and diffusion in discrete time. *PloS One*, 12(4):e0176101, 2017.