BAE - Test plan

The Vaccinators

Cedric De Schepper, Thanh Danh Le,
Wannes Marynen & Stijn Vissers

March 2019

Contents

1 Unit tests

1.1 Daycare & PreSchool tests

For the new data types, we will add tests similar to the ones for the old data types. Since these tests are very much alike for both data types, we will differentiate in the ”generator” and ”populator” tests. These tests will be nearly identical for both data types.

1.1.1 Generator tests

- (a) **OneLocationTest**
  (b) Tests whether a single location will generate enough pools for the population count of that type.
  (c) This is done by adding a single location to a geogrid and applying the correct data type generator to that grid. We can then compare the amount of the pools of that location with the pools of the entire geogrid.
- (a) **ZeroLocationTest**
  (b) Tests whether adding no locations will generate an empty geogrid.
  (c) This is done by applying a generator to the geogrid without adding a location. We can then check if the geogrid is empty.
- (a) **FiveLocationTest**
  (b) Tests whether 5 locations will generate the correct amount of pools for each location.
  (c) This is done by adding 5 different locations to a geogrid. We can then check if every location contains the expected amount of pools.

1.1.2 Populator tests

- (a) **NoPopulation**
  (b) Tests whether having no population won't error.
  (c) This is done by adding a location with a population of 0 to a geogrid. Applying the populator to the grid should not throw an exception.
- (a) **OneLocationTest**
  (b) Tests whether the pools of a single location get populated properly.
  (c) This is done by populating the pools of a geogrid containing one location. Then the capacity of the pools is checked as well as the pools the population is in.
- (a) **TwoLocationTest**
  (b) Tests whether the pools of multiple locations get populated properly.
  (c) This is done by populating a geogrid that contains multiple locations. Then we test whether every person is part of the right pool.

1.2 Data formats tests

1.2.1 JSON

GeoGridJSONReaderTest

- (a) **locationsParsedCorrectlyTest**
  (b) Check whether the JSONReader can read different locations.
  (c) Read some locations from a file, then check whether or not they are present in the geogrid.

- (a) **commutesParsedCorrectlyTest**
  (b) Check whether the JSONReader can read different commutes to different cities.
  (c) Read some locations from a file, then read the commutes to other locations and check whether or not they are present in the geogrid.

- (a) **contactPoolsParsedCorrectlyTest**
  (b) Check whether the JSONReader can read different contactpools.
  (c) Read some contactpools in a city from a file, then check whether or not they are present in the geogrid.

- (a) **peopleParsedCorrectlyTest**
  (b) Check whether the JSONReader can read/parse a person.
  (c) Read a person from a file, then check whether or not they are present in the contactpool of the city.

- (a) **integersParsedCorrectlyTest**
  (b) Check whether the JSONReader can read/parse a person when numericals are properly formatted.
  (c) Similar to the previous test, but this time, numericals aren't formatted as strings in the JSON file.

- (a) **emptyStreamCorrectExceptionTest**
  (b) Check whether the right exception is thrown when you try to parse an empty filestream.
  (c) Try to read a geogrid from an empty string (as a filestream), then check if the exception thrown matches the expected one.

- (a) **invalidTypeCorrectExceptionTest**
  (b) Check whether the right exception is thrown when a contactcenter has a wrong type.
  (c) Try to read a contactcenter with a faulty type, then check if the exception thrown matches the expected one.

- (a) **invalidPersonCorrectExceptionTest**
  (b) Check whether the right exception is thrown when a person's ID that doesn't exist is parsed in a contactpool.
  (c) Try to read a contactpool with a faulty person's ID, then check if the exception thrown matches the expected one.

- (a) **invalidJSONCorrectExceptionTest**
  (b) Check whether the right exception is thrown when trying to read a faulty JSON.
  (c) Try to read a faulty JSON, then check if the exception thrown matches the expected one.

GeoGridJSONWriterTest

- (a) **locationsWrittenCorrectlyTest**
  (b) Check whether the writer can create a correct JSON representing locations.
  (c) Create a geogrid with several locations, then write them to a stream, and check whether it compares to a reference file.

- (a) **commutesWrittenCorrectlyTest**
  (b) Check whether the writer can create a correct JSON representing commutes.
  (c) Create a geogrid with several locations and commutes in between them, then write them to a stream, and check whether it compares to a reference file.

- (a) **contactCentersWrittenCorrectlyTest**
  (b) Check whether the writer can create a correct JSON representing contactcenters.
  (c) Create a geogrid with several locations with contactcenters, then write them to a stream, and check whether it compares to a reference file.

- (a) **peopleWrittenCorrectlyTest**
  (b) Check whether the writer can create a correct JSON representing a population.
  (c) Create a geogrid with several people, then write it to a stream, and check whether it compares to a reference file.

HouseholdJSONReaderTest

- (a) **householdParsedCorrectlyTest**
  (b) Check whether the reader can parse a household file.
  (c) Read a household set from a string, and check whether the reference set in the geogrid contains it.
- (a) **invalidJSONCorrectExceptionTest**
  (b) Check whether the right exception is thrown when trying to read a faulty JSON.
  (c) Try to read a faulty JSON, then check if the exception thrown matches the expected one.
- (a) **emptyStreamCorrectExceptionTest**
  (b) Check whether the right exception is thrown when you try to parse an empty filestream.
  (c) Try to read a referential household from an empty string (as a filestream), then check if the exception thrown matches the expected one.

1.2.2 HDF5

GeoGridHDF5Reader

- (a) **locationsParsedCorrectlyTest**
  (b) Test if reading a location from a hdf5 file is done correctly.
  (c) Read a specific location from a hdf5 file and compare it with what is expected.
- (a) **commutesParsedCorrectlyTest**
  (b) Test if commute info in a hdf5 file, is correct.
  (c) Read a hdf5 file and check if the commute info matches the expected info in the geogrid.
- (a) **contactPoolsParsedCorrectlyTest**
  (b) Test if all types of contactpools can be read correctly from a hdf5 file.
  (c) Specify all different types of contactpools in a single hdf5 file. Read this file and check if all contactpools are present and correct.
- (a) **peopleParsedCorrectlyTest**
  (b) Test if a person can be read correctly from a hdf5 file.
  (c) Read a specific person from a hdf5 file. Compare it with what is expected.

- (a) **invalidTypeCorrectExceptionTest**
  (b) Test if things will be properly handled (exceptions etc.) when non-existing contact center types are read in,
  (c) Read a hdf5 file that contains a non-existing contact center. Check if types of contact center exist.
- (a) **invalidPersonCorrectExceptionTest**
  (b) Test if all the info of a person is present in a hdf5 file.
  (c) Read a hdf5 file that contains at least 1 invalid person. Check if all the needed info of a person is correct and valid.
- (a) **missingLocationsGroupCorrectExceptionTest**
  (b) Check whether the right exception is thrown when you try to parse an a file with missing locations.
  (c) Try to read a hdf5 household that misses locations, then check if the exception thrown matches the expected one.
- (a) **missingPersonsDataSetCorrectExceptionTest**
  (b) Check whether the right exception is thrown when you try to parse an a file with missing persions.
  (c) Try to read a hdf5 household that misses persons, then check if the exception thrown matches the expected one.

GeoGridHDF5Writer

- (a) **locationsWrittenCorrectlyTest**
  (b) Test if location output in hdf5 format is correct and valid.
  (c) Generate a specific location and write it to a hdf5 file. Compare the hdf5 file with the expected hdf5 file.
- (a) **commutesWrittenCorrectlyTest**
  (b) Test if commute info written to a hdf5 file, is done correctly.
  (c) Write specific commute info to a hdf5. Compare the hdf5 file with the expected hdf5 file.
- (a) **contactCentersWrittenCorrectlyTest**
  (b) Test if writing contact centers to a hdf5 file is done correctly.
  (c) Write all the different types of contact centers to a hdf5 file, Compare the hdf5 file with the expected hdf5 file.
- (a) **peopleWrittenCorrectlyTest**
  (b) Test if writing a specific population to a hdf5 file is done correctly.
  (c) Create a specific population and write it to a hdf5 file. Compare the hdf5 file with the expected hdf5 file.

1.3 Data visualisation tests

Since we are not going to test the visual part of this feature, we can only test the data part. This will consist of running some scenario's, and checking whether the "epi-input" and "epi-output" is correctly given by the readers/writers for each of the formats.

- (a) **readerTests**
  (b) For each of the different format readers, we are going to test whether or not they are successful in parsing the information from the "epi-input".
  (c) Read the input for each data format, then check whether our data structure representing each time step contains the info from the input.

- (a) **writerTests**
  (b) For each of the different format readers, we are going to test whether or not they are successful in writing the information to an "epi-output" file.
  (c) Write the output for each data format, then check whether the output is conform with the reference files for the different formats.

1.4 Demographic profile tests

For this feature, we are going to produce different datasets for the provinces, and different sets for central cities versus other locations. To test this feature, we will check if stride can actually read these different input files.

- (a) **provincesParamsTest**
  (b) Check if stride can differentiate between different regions.
  (c) Change the parameters for different regions, and check if the geogrid contains the correct amount of contactpools for each of the provinces.

- (a) **centralCitiesParamsTest**
  (b) Check if stride can read differentiate central cities versus other cities.
  (c) Change the parameters for central cities vs normal cities, and check if the geogrid contains the correct amount of contactpools in central cities vs other cities.

- (a) **HouseholdsParsedCorrectlyTest**
  (b) Check if stride can correctly parse household information.
  (c) Let the householdparser parse an older and a younger reference household set and check wether the parameters are set accordingly.

1.5 Workplace size tests

For this feature, we will be creating a new, adapted workplace reader, generator and populator. Naturally, we will be testing those three things for all the different formats.

1.5.1 workplaceCSVReaderTests

- (a) **fileParsedCorrectlyTest**
  (b) Check whether the reader can read the the workplace sizes from an input file.
  (c) Read the worplace sizes from a file, and check if the sizes of the actual workplaces are conform to the input file.
- (a) **invalidStreamCorrectExceptionTest**
  (b) Check whether the right exception is thrown when you try to parse an an invalid CSV file.
  (c) Try to read a faulty CSV (as filestream), then check if the exception thrown matches the expected one.
- (a) **invalidTypeCorrectExceptionTest**
  (b) Check whether the right exception is thrown when you try to parse an an CSV file with an invalid value type.
  (c) Try to read a CSV (as filestream) containing an invalid value type, then check if the exception thrown matches the expected one.
- (a) **emptyStreamHandledCorrectlyTest**
  (b) Check whether the reader doesn't set any values.
  (c) Try to read a referential workplace size distribution from an empty string (as a filestream), then check if the reader doesn't set any values so the default workplace algorithm holds.

1.5.2 workplaceGeneratorDistributionTests

- (a) **ZeroLocationTest**
  (b) Tests whether there will be any workplaces in an empty geogrid.
  (c) This test will be performed by first creating an empty geogrid, thus with no locations. Afterwards we will check if the number of total workplaces generated will be equal to zero.
- (a) **NoCommuting**
  (b) Tests whether a situation without commutes is generated correctly.
  (c) This test will be performed by first creating a geogrid with several locations with commuting set to 0. We will then check if the amount of workplaces populated is adequate for the amount of people present.

- (a) **NettoNullCommuting**
  (b) Tests whether a situation with as many commutes from A to B as from B to A is generated correctly.
  (c) This test will be performed by first creating a geogrid with several locations with commuting set so the amount of people outgoing equals the amount incoming. We will then check if the amount of workplaces populated is adequate for the amount of people present.
- (a) **TenPercentCommuting**
  (b) Tests whether a situation with ten % of the population commuting is generated correctly.
  (c) This test will be performed by first creating a geogrid with several locations with commuting set to 10%. We will then check if the amount of workplaces populated is adequate for the amount of people present.

1.5.3 workplacePopulatorTests

- (a) **NoCommuting**
  (b) Tests whether workplaces are populated correctly when there is no commuting.
  (c) Given the generated workplaces and population, let the populator run and check if the amount of workplaces of every type matches the expected values.
- (a) **HalfCommuting**
  (b) Tests whether workplaces are populated correctly when half of the working population commutes.
  (c) Given the generated workplaces and population, let the populator run and check if the amount of workplaces of every type matches the expected values.
- (a) **OnlyCommuting**
  (b) Tests whether workplaces are populated correctly when the entire working population commutes.
  (c) Given the generated workplaces and population, let the populator run and check if the amount of workplaces of every type matches the expected values.

1.6 Simulation for Belgium

We believe that for this feature, no specific tests can be written. This is due to fact that we will be using previously implemented features, just with different data. There is no way to test if the data is correct, except for using our common sense when comparing the specific input data versus the general data we used before.

2 Scenario tests

Our scenario tests will consist of running the three basic scenario's of how the stride simulator will be used.

2.1 Default scenario

- (a) **runDefaultScenarioTest**
  (b) Run the default scenario.
  (c) Check whether the simulator runs normally given the default configuration. This means checking that it doesn't throw any errors and exits normally.

2.2 Default generator scenario

- (a) **runGenerateDefaultScenarioTest**
  (b) Run the default generator scenario.
  (c) Check whether the simulator runs normally given the default generator configuration. This means checking that it doesn't throw any errors and exits normally.

2.3 Default import scenario

- (a) **runImportDefaultScenarioTest**
  (b) Run the default import scenario.
  (c) Check whether the simulator runs normally given the default import configuration. This means checking that it doesn't throw any errors and exits normally.