Stride Test & Documentation Plan

Andrei Bondarenko, Mathias Ooms, Stan Schepers, Laurens Van Damme

March 2019

Contents

1 Introduction

In this document you can find an overview of who is responsible for which tasks, how we will evaluate each other and the planned tests for the features that will be implemented by our group, the Striders.

2 Document Plan

2.1 Documentation

Our document plan consists of three stages and is quite simple:

1. Task is carried out and accompanying documentation is written.
2. Documentation and execution/implementation is reviewed by at least one team member.
3. Based on feedback after step 2:
   (a) The task is completed.
   (b) Implement feedback and return to step 2.

2.2 Code commenting

Code comments are added by the creator of the code. This way the creator can explain what he is doing and what he wants to achieve.

2.3 Paper

The person responsible for the paper is Stan Schepers. He will see that everyone does what he has to do and also check the answers that were given. All members are involved divided over the subjects, Stan only makes sure everything is acceptable.

- Introduction: Andrei and Laurens
- Simulation: Andrei and Laurens
- Population generation: Mathias
- Performance profiling: Stan
- Conclusion: Stan

2.4 User manual

The user manual will be the responsibility of Mathias Ooms. Again the whole group will work on it, but he will make sure everything is clear and working. Every team member writes a user manual for the code he wrote. The user manual will then be presented in group to see if everybody can understand. If so, the user manual has been approved.

3 Testing Plan

Tests are made the same way as the user manual. Every member writes tests for his own code and afterwards the whole team will evaluate.

3.1 Daycare & PreSchool - Mathias

3.1.1 DaycareGeneratorTest

- *OneLocationTest*: Check that generator can handle one Location.
  By presetting the GeoGridConfig values and creating a Location for the GeoGrid. On this GeoGrid we apply the generator and verify the Center- and Pool sizes.
- *ZeroLocationTest*: Check that generator can handle empty GeoGrid.
  By presetting the GeoGridConfig values and creating an empty GeoGrid. On this GeoGrid we apply the generator and verify the Center- and Pool sizes.
- *FiveLocationsTest*: Check that generator can handle five Locations.
  By presetting the GeoGridConfig values and creating five Locations for the GeoGrid. On this GeoGrid we apply the generator and verify the Center- and Pool sizes.

3.1.2 PreSchoolGeneratorTest

- *OneLocationTest*: Check that generator can handle one Location.
  By presetting the GeoGridConfig values and creating a Location for the GeoGrid. On this GeoGrid we apply the generator and verify the Center- and Pool sizes.

- *ZeroLocationTest*: Check that generator can handle empty GeoGrid.
  By presetting the GeoGridConfig values and creating an empty GeoGrid. On this GeoGrid we apply the generator and verify the Center- and Pool sizes.

- *FiveLocationsTest*: Check that generator can handle five Locations.
  By presetting the GeoGridConfig values and creating five Locations for the GeoGrid. On this GeoGrid we apply the generator and verify the Center- and Pool sizes.

3.1.3 DaycarePopulatorTest

- *NoPopulation*: Check that Populator can handle empty GeoGrid.
  By creating a GeoGrid without setting it up with ContactCenters. The Populator shouldn't throw any exceptions while using this GeoGrid.

- *OneLocationTest*: Check that Populator can handle one Location.
  By creating a GeoGrid with Daycare's and an example Population and verifying the Populator's distributions with predetermined values (poolId and poolSize, personId and poolId).

- *TwoLocationTest*: Check that Populator can handle two Locations.
  By creating a GeoGrid with Daycare's and an example Population and verifying the Populator's distributions with predetermined values (personId and poolId, checking the allocated ID's).

3.1.4 PreSchoolPopulatorTest

- *NoPopulation*: Check that Populator can handle empty GeoGrid.
  By creating a GeoGrid without setting it up with ContactCenters. The Populator shouldn't throw any exceptions while using this GeoGrid.

- *OneLocationTest*: Check that Populator can handle one Location.
  By creating a GeoGrid with PreSchools and an example Population and verifying the Populator's distributions with predetermined values (poolId and poolSize, personId and poolId).

- *TwoLocationTest*: Check that Populator can handle two Locations.
  By creating a GeoGrid with PreSchools and an example Population and verifying the Populator's distributions with predetermined values (personId and poolId, checking the allocated Id's).

3.1.5 GeoGridJSONReaderTest

(Currently out of target)

- *contactCentersTest*: Check that JSON input is correctly interpreted.
  By reading in a predetermined file and verifying this with the created objects. (Extended with Daycare & PreSchool)

- *peopleTest*: Check that JSON input is correctly interpreted.
  By reading in a predetermined file and verifying this with the created objects. (Extended with Daycare & PreSchool)

- *intTest*: Check that JSON input is correctly interpreted.
  By reading in a predetermined file and verifying this with the created objects. (Extended with Daycare & PreSchool)

3.1.6 GeoGridJSONWriterTest

(Currently out of target)

- *contactCentersTest*: Check that objects are correctly written to a JSON-file.
  By creating predetermined objects and write these to files. Followed by verifying this with a predetermined file. (Extended with Daycare & PreSchool)

- *peopleTest*: Check that objects are correctly written to a JSON-file.
  By creating predetermined objects and write these to files. Followed by verifying this with a predetermined file. (Extended with Daycare & PreSchool)

3.2 Data formats - Andrei

3.2.1 HouseholdJSONReader

- *emptyFileTest*: Check whether or not the reader can handle an empty file appropriately.
  Feed an empty file to the reader and check if the reader throws the correct exception.

- *noHouseholdTest*: Check whether or not the reader can handle a file with missing *housholdsList* parameter or with an empty *housholdsList*.
  Feed a file containing zero households to the reader and check if the right number of households is instantiated.

- *singleHouseholdTest*: Check whether or not the reader can handle a file containing data for one single household.
  Feed a file containing one household to the reader and check if the right number of households is instantiated.

- *multiHouseholdTest*: Check whether or not the reader can handle a file containing data for multiple households of the same size.
  Feed a file containing multiple households to the reader and check if the right number of households is instantiated.

- *multiHouseholdVarSizeTest*: Check whether or not the reader can handle a file containing data for multiple households of variable sizes.
  Feed a file containing multiple households with different sizes to the reader and check if the right number of households with correct sizes is instantiated.

- *invalidJSONTest*: Check whether or not the reader can handle a file containing invalid JSON.
  Feed a file containing invalid JSON to the reader and check if the right exceptions are thrown.

- *stringFormattedNumbersTest*: Check whether or not the reader can handle a file containing ages (integers) formatted as strings.
  Feed a file containing string formatted integers to the reader and check if no exceptions are thrown and the right conversions made.

3.2.2 GeoGridJSONReader

- *readLocationsTest*: Check whether or not the reader can read properly formatted locations and create appropriate corresponding objects.
  Feed a GeoGrid file containing only locations to the reader and check if the right number of locations is instantiated.

- *readCommutesTest*: Check whether or not the reader can read properly formatted commutes and create appropriate corresponding objects.
  Feed a GeoGrid file containing commutes to the reader and check if the right number of commutes is instantiated.

- *readContactPoolsTest*: Check whether or not the reader can read properly formatted contact pools and create appropriate corresponding objects.
  Feed a GeoGrid file containing contact pools to the reader and check if the right number of contact pools is instantiated.

- *readPeopleTest*: Check whether or not the reader can read properly formatted people and create appropriate corresponding objects.
  Feed a GeoGrid file containing people to the reader and check if the right number of people is instantiated.

- *emptyFileTest*: Check whether or not the reader can handle empty files.
  Feed an empty file to the reader and check if the reader throws the correct exception.

- *invalidTypesTest*: Check whether or not the reader can handle files containing incorrect types for certain attributes.

Feed a file containing incorrect types to the reader and check if the reader throws the correct exceptions.

- *invalidJSONTest*: Check whether or not the reader can handle files containing invalid JSON.
  Feed a file containing invalid JSON and check if the reader throws the correct exception.

3.2.3 GeoGridJSONWriter

- *writeLocationsTest*: Check whether or not the writer produces the expected output when writing locations to a file.
  Write a location to a JSON file and compare with expected output.
- *writeCommutesTest*: Check whether or not the writer produces the expected output when writing commutes to a file.
  Write a commute to a JSON file and compare with expected output.
- *writeContactPoolTest*: Check whether or not the writer produces the expected output when writing contact pools to a file.
  Write a contact pool to a JSON file and compare with expected output.
- *writePeopleTest*: Check whether or not the writer produces the expected output when writing people to a file.
  Write a person to a JSON file and compare with expected output.

3.2.4 GeoGridHDF5Reader

- *readLocationsTest*: Check whether or not the reader can read locations from a valid HDF5-file.
  Feed a valid HDF5 file containing locations to the reader and check if correct objects are instantiated.
- *readCommutesTest*: Check whether or not the reader can read commutes from a valid HDF5-file.
  Feed a valid HDF5 file containing commutes to the reader and check if correct objects are instantiated.
- *readContactPoolTest*: Check whether or not the reader can read contact pools from a valid HDF5-file.
  Feed a valid HDF5 file containing contact pools to the reader and check if correct objects are instantiated.
- *readPeopleTest*: Check whether or not the reader can read people from a valid HDF5-file.
  Feed a valid HDF5 file containing people to the reader and check if correct objects are instantiated.

- *emptyFileTest*: Check whether or not the reader can handle empty files. Feed an empty file to the reader and check if the reader throws the correct exception.

3.2.5 GeoGridHDF5Writer

- *writeLocationsTest*: Check whether or not the writer produces the expected output when writing locations to a file.
  Write a location to an HDF5 file and see if output contains expected information.

- *writeCommutesTest*: Check whether or not the writer produces the expected output when writing commutes to a file.
  Write a commutes to an HDF5 file and see if output contains expected information.

- *writeContactPoolTest*: Check whether or not the writer produces the expected output when writing contact pools to a file.
  Write a contact pool to an HDF5 file and see if output contains expected information.

- *writePeopleTest*: Check whether or not the writer produces the expected output when writing people to a file.
  Write a person to an HDF5 file and see if output contains expected information.

3.3 Data visualization - Laurens

3.3.1 EpiOutputGeneratorTest

- *ZeroMembersTest*: Checks if the function generateEpiOutput of the class Contactpool returns the expected result with no members in the contactpool.
  A contactpool object with zero member objects is created and the function is called. The result will be compared to the right result it should give.

- *OneMemberTest*: Checks if the function generateEpiOutput of the class Contactpool returns the expected result with one member in the contactpool.
  A contactpool object with one member object is created and the function is called. The result will be compared to the right result it should give.

- *FiveMembersTest*: Checks if the function generateEpiOutput of the class Contactpool returns the expected result with five members in the contactpool.
  A contactpool object with five member objects is created and the function is called. The result will be compared to the right result it should give.

- *zeroContactpoolsTest*: Checks if the function generateEpiOutput of the class Location returns the expected result with no contactpools in the location.
  A location object is created with zero contactpool objects. The function will be called and the result will be compared with the right result it should give.

- *oneContactpoolTest*: Checks if the function generateEpiOutput of the class Location returns the expected result with one contactpool in the location.
  A location object is created with one contactpool object. The function will be called and the result will be compared with the right result it should give.

- *fiveContactpoolsTest*: Checks if the function generateEpiOutput of the class Location returns the expected result with five contactpools in the location.
  A location object is created with five contactpool objects. The function will be called and the result will be compared with the right result it should give.

3.3.2 EpiOutputJSONWriterTest

- *zeroLocationsTest*: Checks if the writer creates the expected epi-output JSON-file with the right name and zero locations in the geogrid.
  A geogrid object is created with zero location objects and is then written to a JSON-file with the JSON writer. The output will be compared to the right output it should give.

- *oneLocationZeroDaysTest*: Checks if the writer creates the expected epi-output JSON-file for one location in the geogrid for zero time steps.
  A geogrid object is made with one location object which has zero contactpool objects. Then the geogrid will be written to a JSON-file with the JSON writer. The output will be compared to the right output it should give.

- *oneLocationOneDayTest*: Checks if the writer creates the expected epi-output JSON-file for one location in the geogrid for a one day period.
  A geogrid object is made with one location object which has one or more contactpools with members. Then the geogrid will be written to a JSON-file with the JSON writer. The output will be compared to the right output it should give.

- *TwoLocationsFiveDaysTest*: Checks if the writer creates the expected epi-output JSON-file for two locations in the geogrid for a five day period.
  A geogrid object is made with two location objects which all have one or more contactpools with members. Then the geogrid will be written to

a JSON-file with the JSON writer. The output will be compared to the right output it should give.

3.3.3 EpiOutputJSONReaderTest

- *zeroLocationsTest*: Check if an epi-output JSON-file with zero locations in can be read and converted correctly.
  A JSON-file with zero locations is used and fed to the reader. The reader then converts the locations to the right c++ objects. These are then checked in a comparison with what the reader should have created.

- *oneLocationZeroDaysTest*: Check if an epi-output JSON-file with one location without any time passed can be read and converted correctly.
  A JSON-file with one location is used and fed to the reader. The reader then converts the location to the right c++ objects. This object is then checked in a comparison with what the reader should have created.

- *oneLocationOneDayTest*: Check if an epi-output JSON-file with one location, over a period of one day, can be read and converted correctly.
  A JSON-file with one location and one day is used and fed to the reader. The reader then converts the location to the right c++ objects. This object is then checked in a comparison with what the reader should have created.

- *TwoLocationsFiveDaysTest*: Check if an epi-output JSON-file with five locations, over a period of one day, can be read and converted correctly.
  A JSON-file with two locations, which all have five days, is used and fed to the reader. The reader then converts the locations to the right c++ objects. These are then checked in a comparison with what the reader should have created.

- *emptyStreamTest*: Checks if an exception is thrown when the stream is empty.
  An empty stream is made and putted in the reader. This should cause an exception.

- *invalidJSONTest*: Checks if an exception is thrown when the stream has a invalid JSON-file as input.
  A self made invalid JSON-file is putted in the reader. This will normally cause an exception.

3.3.4 EpiOutputProtoWriterTest

- *zeroLocationsTest*: Checks if the writer creates the expected epi-output protobuf-file with the right name and zero locations in the geogrid.
  A geogrid object is created with zero location objects and is then written to a protobuf-file with the protobuf writer. The output will be compared to the right output it should give.

- *oneLocationZeroDaysTest*: Checks if the writer creates the expected epi-output protobuf-file for one location in the geogrid for zero time steps.
  A geogrid object is made with one location object which has zero contactpool objects. Then the geogrid will be written to a protobuf-file with the protobuf writer. The output will be compared to the right output it should give.

- *oneLocationOneDayTest*: Checks if the writer creates the expected epi-output protobuf-file for one location in the geogrid for a one day period.
  A geogrid object is made with one location object which has one or more contactpools with members. Then the geogrid will be written to a protobuf-file with the protobuf writer. The output will be compared to the right output it should give.

- *TwoLocationsOneFiveTest*: Checks if the writer creates the expected epi-output protobuf-file for two locations in the geogrid for a five days period.
  A geogrid object is made with two location objects which all have one or more contactpools with members. Then the geogrid will be written to a protobuf-file with the protobuf writer. The output will be compared to the right output it should give.

3.3.5 EpiOutputProtoReaderTest

- *zeroLocationsTest*: Check if an epi-output protobuf-file with zero locations in can be read and converted correctly.
  A protobuf-file with zero locations is used and fed to the reader. The reader then converts the locations to the right c++ objects. These are then checked in a comparison with what the reader should have created.

- *oneLocationZeroDaysTest*: Check if an epi-output protobuf-file with one location without any time passed can be read and converted correctly.
  A protobuf-file with one location is used and fed to the reader. The reader then converts the location to the right c++ objects. This object is then checked in a comparison with what the reader should have created.

- *oneLocationOneDayTest*: Check if an epi-output protobuf-file with one location, over a period of one day, can be read and converted correctly.
  A protobuf-file with one location and one day is used and fed to the reader. The reader then converts the location to the right c++ objects. This object is then checked in a comparison with what the reader should have created.

- *twoLocationsFiveDaysTest*: Check if an epi-output protobuf-file with five locations, over a period of five days, can be read and converted correctly.
  A protobuf-file with two locations, which all have five days, is used and fed to the reader. The reader then converts the locations to the right c++

objects. These are then checked in a comparison with what the reader should have created.

- *emptyStreamTest*: Checks if an exception is thrown when the stream is empty.
  An empty stream is made and putted in the reader. This should cause an exception.

- *invalidprotobufTest*: Checks if an exception is thrown when the stream has a invalid protobuf-file as input.
  A self made invalid protobuf-file is putted in the reader. This will normally cause an exception.

3.3.6 EpiOutputHDF5WriterTest

- *zeroLocationsTest*: Checks if the writer creates the expected epi-output HDF5-file with the right name and zero locations in the geogrid.
  A geogrid object is created with zero location objects and is then written to a HDF5-file with the HDF5 writer. The output will be compared to the right output it should give.

- *oneLocationZeroDaysTest*: Checks if the writer creates the expected epi-output HDF5-file for one location in the geogrid for zero time steps.
  A geogrid object is made with one location object which has zero contactpool objects. Then the geogrid will be written to a HDF5-file with the HDF5 writer. The output will be compared to the right output it should give.

- *oneLocationOneDayTest*: Checks if the writer creates the expected epi-output HDF5-file for one location in the geogrid for a one day period.
  A geogrid object is made with one location object which has one or more contactpools with members. Then the geogrid will be written to a HDF5-file with the HDF5 writer. The output will be compared to the right output it should give.

- *twoLocationsFiveDaysTest*: Checks if the writer creates the expected epi-output HDF5-file for two locations in the geogrid for a five day period.
  A geogrid object is made with two location objects which all have one or more contactpools with members. Then the geogrid will be written to a HDF5-file with the HDF5 writer. The output will be compared to the right output it should give.

3.3.7 EpiOutputHDF5ReaderTest

- *zeroLocationsTest*: Check if an epi-output HDF5-file with zero locations in can be read and converted correctly.
  A HDF5-file with zero locations is used and fed to the reader. The reader then converts the locations to the right c++ objects. These are then checked in a comparison with what the reader should have created.

- *oneLocationZeroDaysTest*: Check if an epi-output HDF5-file with one location without any time passed can be read and converted correctly.
  A HDF5-file with one location is used and fed to the reader. The reader then converts the location to the right c++ objects. This object is then checked in a comparison with what the reader should have created.

- *oneLocationOneDayTest*: Check if an epi-output HDF5-file with one location, over a period of one day, can be read and converted correctly.
  A HDF5-file with one location and one day is used and fed to the reader. The reader then converts the location to the right c++ objects. This object is then checked in a comparison with what the reader should have created.

- *twoLocationsFiveDaysTest*: Check if an epi-output HDF5-file with five locations, over a period of five days, can be read and converted correctly.
  A HDF5-file with two locations, which all have five days, is used and fed to the reader. The reader then converts the locations to the right c++ objects. These are then checked in a comparison with what the reader should have created.

- *emptyStreamTest*: Checks if an exception is thrown when the stream is empty.
  An empty stream is made and putted in the reader. This should cause an exception.

- *invalidHDF5Test*: Checks if an exception is thrown when the stream has a invalid HDF5-file as input.
  A self made invalid HDF5-file is putted in the reader. This will normally cause an exception.

3.4 Demographic profile - Stan

3.4.1 GeoGridConfig

- SetDataTest: Check if the *SetData* function can handle multiple household files.

- SetDataInputTest: Check if *SetData* sets correctly the values of *input* in the GeoGridConfig for multiple household files.

- SetDataRefHHTest: Check if *SetData* sets correctly the values of $refHH$ in the GeoGridConfig for multiple household files.

- SetDataPopInfoTest: Check if *SetData* sets correctly the values of $popInfo$ in the GeoGridConfig for multiple household files.

- SetDataPoolsTest: Check if *SetData* sets correctly the values of *pools* in the GeoGridConfig for multiple household files.

3.4.2 HouseholdPopulatorTest

- PopulationIdNotInConfigTest: test if an exception is thrown if a location has no matching household reference.
- GetHouseHoldFilePerPopulationIdTest: Check of the function gets all info from the property tree.

3.4.3 CitiesCSVReader

- *test*1: Change test with population_id and test if population_id is set correctly.

3.4.4 GeoGridIOUtils

- *compareLocation*: Add check for population_id.

3.4.5 HouseHoldCSVReader

- $MultipleHouseholdsTest$: Check if multiple household files can be set correctly in the GeoGridConfig.
  We're testing if the the *population_id* are set correctly.

3.4.6 HouseHoldPopulator

- $HouseholdRightRefHHTest$: Test if the right household reference is used for drawing the ages.
  We test this by creating two reference households with two different ages and checking if the ages of the households match with the given reference household in the file with the cities.

3.5 Big class personnel of companies - Mathias

3.5.1 WorkplaceFormatReaderTest

- $ReaderTestX$: Check that Reader can handle the associated formats.
  By creating input values in the according format, passing this input into the Reader and comparing the acquired values from the reader with the original input.

3.5.2 WorkplaceGeneratorTest

- $ZeroDistributionTest$: Check that generator can handle empty distribution with multiple Locations.
  By presetting the GeoGridConfig values, creating a GeoGrid with multiple Locations, but an empty distribution. On this GeoGrid we apply the generator and verify the Pool sizes/limits.

- *DistributionTest*: Check that generator can handle distribution with multiple Locations.
  By presetting the GeoGridConfig values, creating multiple Locations for the GeoGrid. On this GeoGrid we apply the generator and verify the Pool sizes/limits.

3.5.3 WorkplacePopulatorTest

- *ZeroDistributionTest*: Check that populator can handle empty distribution.
  By presetting the GeoGridConfig values, creating a GeoGrid with multiple Locations, but an empty distribution. On this GeoGrid we apply the populator and verify the Pool sizes/limits.

- *DistributionTest*: Check that populator can handle distribution.
  By presetting the GeoGridConfig values, creating multiple Locations for the GeoGrid. On this GeoGrid we apply the populator and verify the Pool sizes/limits.

3.5.4 WorkplacePopulatorTest

- *ZeroDistributionTest*: Check that populator can handle empty distribution.
  By presetting the GeoGridConfig values, creating a GeoGrid with multiple Locations, but an empty distribution. On this GeoGrid we apply the populator and verify the Pool sizes/limits.

- *DistributionTest*: Check that populator can handle distribution.
  By presetting the GeoGridConfig values, creating multiple Locations for the GeoGrid. On this GeoGrid we apply the populator and verify the Pool sizes/limits.

3.5.5 ScenarioRuns

- *geopop_distribution*: Check effect on simulations with distributions.
  By using the existing target values, creating a special distribution to approximate the existing results.