Documentation for the framework. Report for the second task.

Description

The developed application, requested in the second evaluation task of the Allpix Squared project, serves as an example of usage of multithreading in C++ 11/14. The framework incorporates several tools for handling concurrent access to the data, emulating simulation of particles passing through a detector setup, multithreaded processing of separate events/tasks, ordered reporting on simulated events, etc. Four dummy Modules are implemented for emulation of an event. The software uses several Mersenne Twister pseudo random number generators (PRNG).

Source code

The source code of the developed framework is available in `src` folder of my GitHub repository devoted to the GSoC 2019 project:
https://github.com/Mikhail-Vaganov/gsoc_2019_allpix_squared

Build

For building the framework, only the standard libraries are required. The program has been tested on MacOS X El Capitan 10.11.6 and on Ubuntu 18.04.2.

Linux: An example command for building on Linux system, providing the current working directory is `src`:

```
:~$ g++  ./*.cpp  -o framework -std=c++11 -pthread
```

Instead of "`framework`" one can type any other desirable name of the output executable.
Option `-pthread` may help if the multithreading mode is unavailable by default.

MacOS: an analogous instruction can be executed on this system:

```
:$ g++  ./*.cpp -o framework -std=c++11
```

Command line variables

The built executable can be run with up to three optional arguments:

```
:$ ./framework 10000 4 2364
```

The first one is the number of events that should be simulated, the second variable represents the number of workers in the inner thread pool handling the queue of events, and the last variable is a seed for the main PRNG. The default value of each of these arguments is 1.

Setting up modules

During simulation of an event, several modules are executed, their output messages are collected and transferred to a Reporter entity, which, in turn, streams the result into `std::cout`. The set of the modules is hardcoded in ModuleSet class and can be tuned via changing the constructor in ModuleSet.cpp file.

Output

Running the framework without arguments would simulate one event using one worker, and the main PRNG would be seeded with 1. In this case, a possible output will look like this:

```
Deposition      2034626415      241251864
Propagation     2034626415      241251864
Transfer        2034626415      241251864
Digitizer       2034626415      241251864
```

In the current implementation PRNGs of all the Modules in one Event are seeded with the same value created by the main PRNG. Different Events receive different integers generated by the main PRNG.