1. ARCHITECTURE EVALUATION:

[A1] **Architecture violation for module projet:** The module **projet** only handles the command line arguments (argc and argv) and only connect with the Model sub-system through the **simulation** module. Only a function or method of this module has to be called for rendu1.

[A2] **Architecture violation for the Model sub-system:** The simulation module has to supervise other modules => it includes the interface of other modules. The interface simulation.h must NOT be included in other modules of the Model or in squarecell. Likewise other high-level modules must not be included in lower levels modules (fig 9a).

It is allowed to introduce inter-dependencies between modules of the same layer, in case there are more than one module to represent different types of ants

It is allowed to have more modules compared to Fig 9a but it should be justified somewhere in the headers and the group should have the authorization from the lecturer for having additional modules

[A3] **Architecture violation for module squarecell:** The module **squarecell** has to be independent from higher level modules, as described on p11 section 7.3.1 "Indépendance de squarecell":

AUCUN des noms de types/concepts de Tchanz ne doit apparaître dans **squarecell**

Ex : including the appendix A = « constantes.h » in the squarecell module is a clear violation of the architecture specification.

,

The spreadsheet column L shows the **default maximum of 3 points** for ARCHITECTURE.

=> **Remove 1 point per violation** (max 1 pt for each, max 3pt in total) in that column.

In the spreadsheets' column M architecture violation_comment, note down the corresponding **code [A1], [A2], [A3]** together with a short justification.

4. CLASS ENCAPSULATION / MODULARIZATION:

[C0] Incomplete implementation: the assignment requires to read a configuration file to perform the detection of errors. Hence the max number of points is reduced if this task is only partially implemented. ***The case has to be reported to RB who will have a look and calibrate the reduced max.***

[C1] Encapsulation violation : using any **global variable** or making any **attribute public** is strictly forbidden in any modules, including **public** static attributes (no problem for methods and static methods).

It is allowed to have static variables in the implementation (.cc) of a module or variables declared in the unnamed namespace, or **private** static attribute (=> warning if there are too many of them).
=> BIG warning in case some static variables appear in the interface of a module.

[C2] Externalization of methods' definition : whenever a module interface shows a class interface, it should contain only method prototypes. The method definition must be externalized in the module implementation.

The only *accepted exception* of method definition in the class interface are the **constructors** or **getters** methods that fits onto the same line as the function prototype.

The spread sheet column N shows the **default maximum of 3 points** for **CLASS**.

=> **Remove 1 point per public attribute or global variable** (max 3pt).

=> **Remove 1 point per interface that is not correctly externalized** (max 3 pt).

The total of removed points from C1 and C2 is maximum 3 pts.

In the spreadsheet column O class violation_comment, note down the corresponding **code [C1],[C2]** together with the **interface name** and the **public attribute name**. Indicate that it must be corrected in future assignments.

5. CODING STYLE

[L1] Indentation rules have been ignored **more than 4 times** ; read carefully the conventions before considering this penalty because we accept some variants. Please note that we don't indent the public/private keywords in class declaration. Indicate only a **warning** if the whole code is consistent in the use of multiple brace styles (e.g. two styles are used but always in the same way, for the same control instructions)

[L2] There are **more than 4 wrapping line** in the code (more than 87 char); Indicate only a warning if 4 wrapping lines or less.

[P2] Apart from a single function of max 80 lines, all function size must not exceed 40 lines (+tolerance of 2 lines) with geany (with the default font size).

[P5] 1) there are more than 4 missing symbols or constexpr variables: Using a symbol or a constexpr variable in place of a raw numeric value in the code is a good practice in the following cases: if this numeric value is a parameter that may change as the program evolves or if it is an approximation of a math/physics constant (e.g. Pi). The chosen symbol has to convey the meaning of the parameter or information it represents.
Apart from these cases we should leave the raw numeric values in the code. A clear example is: numeric values resulting from solving an equation because there is no point of changing these values, they are not "parameters".

[P5] 2) there are more than 4 poor choice of symbol name such as ZERO, ONE, TWO… respectively for 0, 1, 2… Such choices show that the person has not understood that the purpose of a symbol or a constexpr variable is to convey the MEANING of the numeric value. It is allowed to define a symbol for Pi and similar constants because it is important to use the same approximation throughout the code and it would be tedious and error-prone to write such approximation explicitly.

The spreadsheet column P shows the default maximum of 4 points for STYLE

=> **remove 1 point per coding style criteria that is violated**

In the spreadsheet column R violation_list, note down the **code** representing the violated criteria followed by the **filename** and the **line number** it occurs. For instance **[L2]simulation.cc57,65,80-84** means that this set of lines are violating the wrapping criteria in the file simulation.cc. If the same type of violation occurs more than 5 times, you mention briefly how much larger the problem is in the violation comment column R

Keep the violation_list alphabetically sorted and separate each entry by a comma.