1. ARCHITECTURE EVALUATION:

[A1] <u>Architecture violation for module projet</u>: 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 of this module has to be called for rendu1.

[A2] <u>Architecture violation for the Model sub-system</u>: 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 tools.

It is allowed to introduce inter-dependencies between player, ball and map.

It is allowed to have more modules but it should be justified somewhere in the headers ; indicate a **warning** to ask authorization to the lecturer for having additional modules

[A3] <u>Architecture violation for module tool</u>: The module **tool** has to be independent from higher level modules as described on p6 "contrainte sémantique importante" of the project description. I cite it here:

> Contrainte sémantique importante : par construction, étant de bas-niveau, le module tool ne doit connaitre que des entités génériques telles que points, vecteurs, segments de droite, cercles et carrés dans le plan. Ce module doit être indépendant des entités du jeu afin d'être ré-utilisable ultérieurement dans d'autres applications. Cela veut dire que le module tool n'a aucune idée de ce qu'est un joueur, une balle ou un obstacle: CES MOTS/CONCEPTS NE DOIVENT PAS APPARAITRE dans le module tool. C'est seulement au niveau des appels des fonctions dans les modules player, ball et map que les données fournies en argument seront explicitement des cercles représentant des joueur et des balles ou des carrés représentant des obstacles.

The spreadsheet column "Architecture" shows the **default maximum of 3 points**

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

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

4. CLASS ENCAPSULATION / MODULARIZATION:

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

It is allowed to have static variables in a module (indicate a warning if there are too many of them)

[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 **getters** methods that fits onto the same line as the function prototype.

The spread sheet column "Class encapsul" shows the **default maximum of 4 points**.

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

=> **Penalty=0.5 point per interface that is not correctly externalized** (max 3 pt).

The total of removed points from C1 and C2 is maximum 4 pts.

In the spreadsheet column 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** according to the conventions. 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 two functions of max 80 lines, all function size must not exceed 40 lines (+tolerance of 4 lines) with geany (with the default font size).

[P5] 1) there are more than 4 Missing symbols: Using a symbol 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 justification 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 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 Code Style shows the default maximum of 4 points

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

In the spreadsheet column violation_list, note down the **code** representing the violated criteria followed by the **line number** it occurs. For instance **[L2]57,65,80-84** means that this set of lines are violating the wrapping criteria. 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.

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