Final Report grading by RB **(the pdf has to be provided in the archive file):**

Description des algorithms: 0.75pt

Commentaire sur la progression de la simulation t44 ou t45 avec quelques images: 0.5pt

Méthodologie, auto-évaluation et conclusion: 0.75pt

Execution grading:

ALL test scenarios except 44 & 45 require to reset the **desintegration rate to zero** ; so we edited **constantes.h** before launching **make**.

Column [**Arg] => obtaining an executable with `make`** gives 0.25pts

The other 0.25pt is obtained if **the project can be executed without providing a filename argument on the command line**: **`./projet`**
The drawing area should be empty.

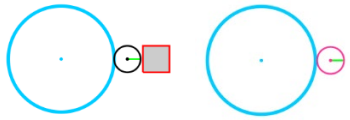
General

- **Assessment of speed (number of simulation steps to perform a test):** The numbers of steps between parenthesis are indicative of the fastest performance that can happen when multiplying speed by delta_t. Some tests include accepted variant behaviors. We also agreed to get slower performance (more simulation steps to reach a goal) because it was allowed to move at a slower speed than the maximum speed. HOWEVER, forgetting to multiply the speed by delta_t makes the movements **8** times faster, hence make all tests to fail because the robots move too fast. Instead of giving 0 to all tests we decided to penalize this mistake globally and try to interpret the test execution without the number of step criteria.
    - Penalty for forgetting to multiply by delta_t in translation : -1pt
    - Penalty for forgetting to multiply by delta_t in rotation : -1pt

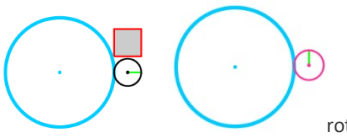
- **Simulation does not stop when the cleaning task is finished**: -0.5pt for t28
- **No color change when a robot is colliding a particle**: -0.5pt for t32
    - But no penalty if the particle is destroyed within the same update
    - Penalty only if there is no color change during successive colliding updates

We expanded the application window to see clearly what happens.

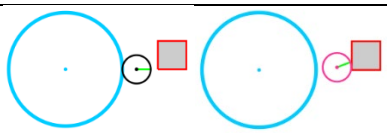
Column **t28** : **test type0 in straight line (0.5pt)** cf Rendu3 section 2

Le robot neutraliseur de **type0** se dirige vers la particule qu'il atteint en 2 mises à jour. La particule est supprimée et le robot revient vers le robot spatial Fin de la simulation lorsque le robot est revenu dans la robot spatial (213 steps pour le prog demo)	 2 steps

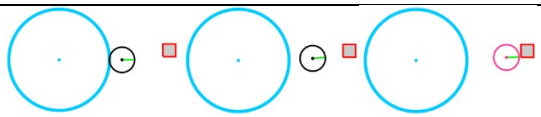
Column **t29** : **test type0 (1pt)** cf Rendu3 section 2

Le robot neutraliseur de type0 tourne d'abord sur place (99) avant de se diriger vers la particule qu'il atteint en 2 mises à jour La particule est supprimée et le robot revient vers le robot spatial Fin de la simulation lorsque le robot est revenu dans la robot spatial (211)	 rot +2 steps

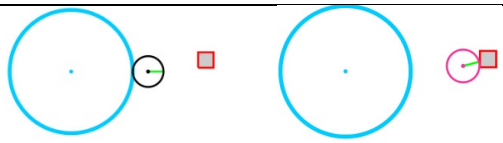
Column **t30** : **test type0 (1pt)** cf Rendu3 section 2

Comportement A) Le robot neutraliseur de type0 tourne d'abord sur place (23) avant de se diriger vers la particule qu'il atteint en 5 mises à jour puis il tourne sur place pour s'aligner sur la normale à la particule (51). La particule est supprimée et le robot revient vers le robot spatial (262). Fin de la simulation lorsque le robot est revenu dans la robot spatial. Comportement B) cas limite où on accepte que le robot avance tout droit car déjà perpendiculaire face particule	 Comportement A: rot +5 steps suivi par rot pour être perpendiculaire et éliminer la particule

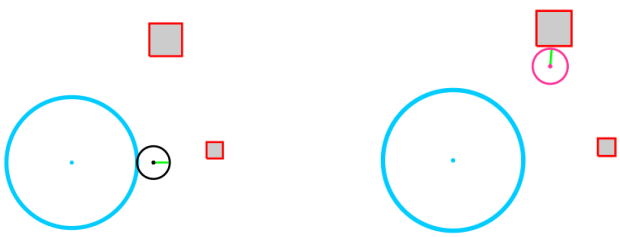
Column **t31** : **test type1 (1pt)** cf Rendu3 section 2

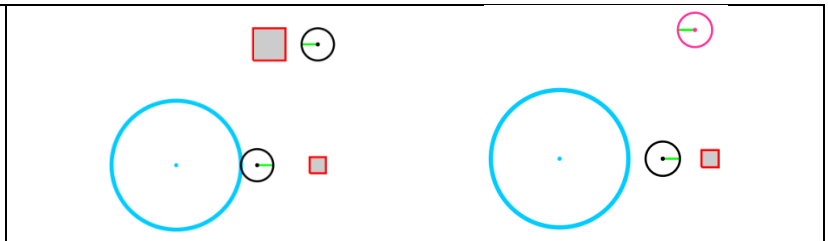
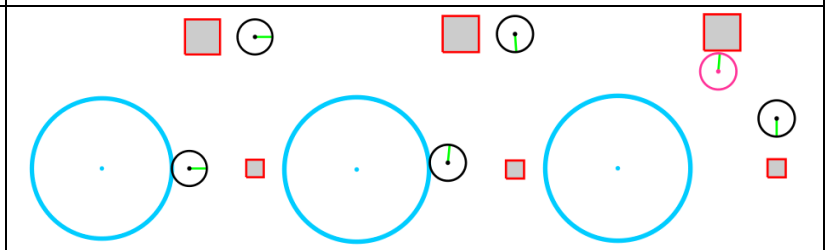
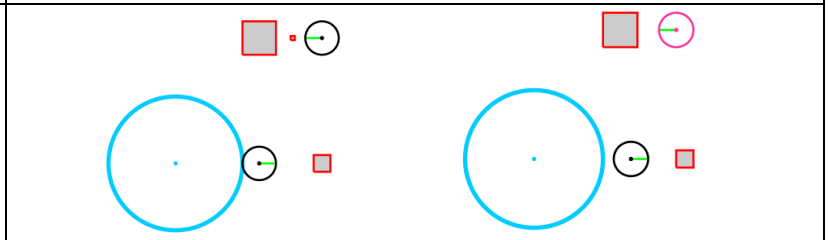
Le robot neutraliseur de type1 tourne d'abord sur place avant (11) de se diriger vers la particule (17), en restant en dehors de la zone à risque, puis il tourne sur place pour s'aligner sur la normale à la particule (24), et enfin il avance vers la particule (37-39). La particule est supprimée et le robot revient vers le robot spatial. Fin de la simulation lorsque le robot est revenu dans la robot spatial. Il peut viser un point different de celui visible ici du moment qu'il est en dehors de la zone à risque et face à une face du carré de la particule.	

Column **t32** : **test type2 (1pt)** cf Rendu3 section 2

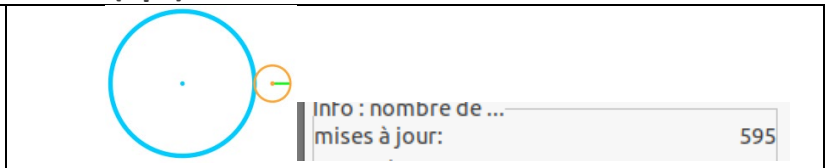
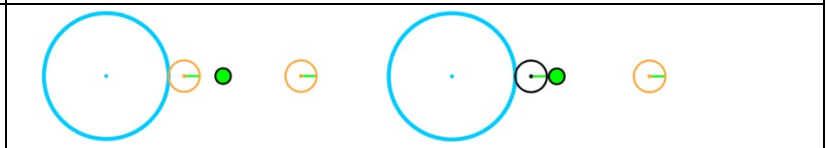
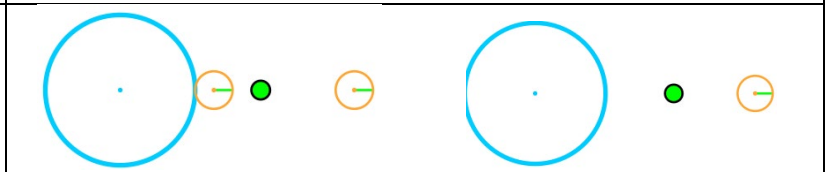
Le robot neutraliseur de type2 avance en tournant jusqu'au contact (19) et finit de s'aligner sur la normale à la particule quand il est en collision avec celle-ci. La particule est supprimée (35) et le robot revient vers le robot spatial. Fin de la simulation lorsque le robot est revenu dans la robot spatial (232)	

Column **[part]** : **tests particules (2pt)** cf Rendu3 section 2

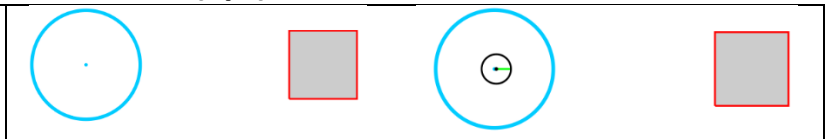
t33: La particule la plus grande doit être choisie comme cible même si elle est plus éloignée, contact vers (137) ; elimination vers (142) ensuite seulement la petite est traitée, contact vers (334) ; elimination vers (340), puis retour et fin vers (488) ⇨ 0.5pt	

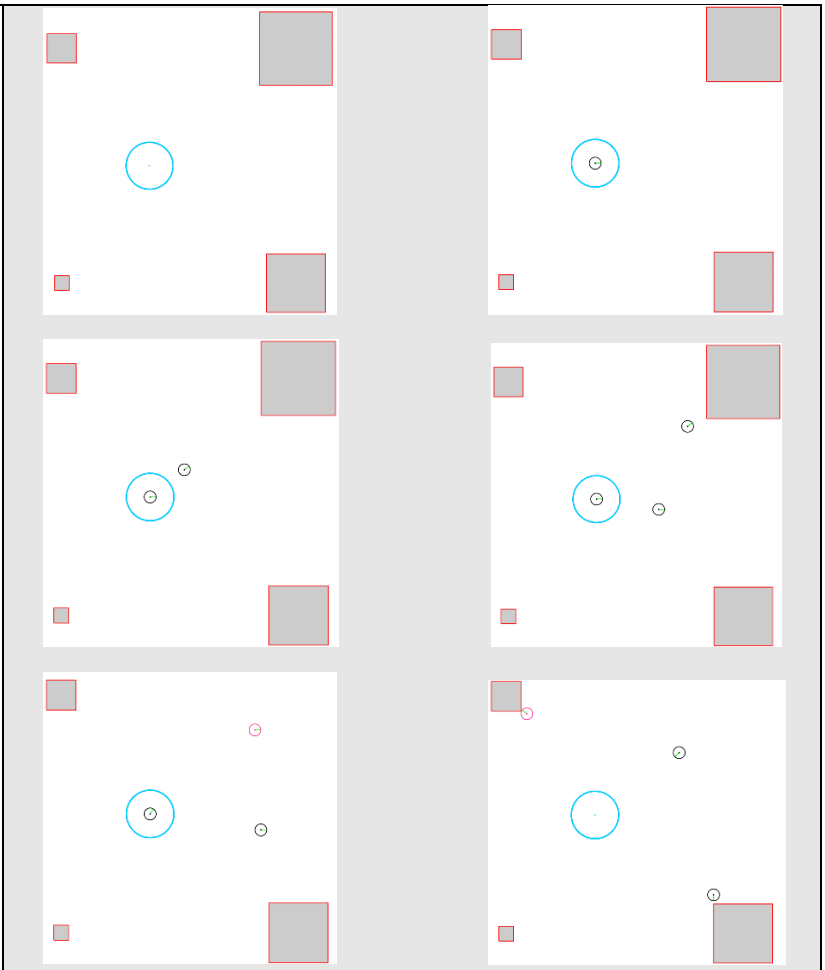
t34 et t35: La particule la plus grande doit choisir le robot le plus proche; ensuite la particule suivante choisit le robot restant le plus proche. Ici pas d'ambiguïté. **t35 :** l'ordre des robots est inversé dans le fichier ⇨ 2 x 0.25pt	
t36: La particule la plus grande doit choisir le robot le plus proche ; dans ce cas la difference d'orientation impacte le choix du robot le plus proche. C'est le robot du bas qui est le plus "proche" en temps de parcours de la grande particule. ⇨ 0.5pt	
t37: Une particule se trouve sur le chemin de la particule la plus grande ; elle doit être détruite par le robot choisi pour la particule la plus grande (5). ⇨ 0.5pt	

Column **[rep]** : **tests panne & réparations (2pt)** cf Rendu3 section 2

t38: Un robot neutraliseur en panne est détruit après 5 mises à jour car cela correspond à la durée max de 600 depuis qu'il est tombé en panne ⇨ 0.5pt	
t39: le robot réparateur se dirige vers le robot neutraliseur en panne le plus proche. ⇨ 0.5pt	
t40: le robot réparateur se dirige vers le robot neutraliseur en panne le plus proche MAIS seulement s'il reste suffisamment de temps au robot réparateur pour l'atteindre ; ici il se dirige donc vers l'autre robot neutraliseur en panne *dès le début.* => 0.5pt **t41:** le réparateur et les neutraliseurs rentrent car aucune panne ni particule => 0.5pt	

Column **[crea]** : **tests creation de neutraliseurs (1pt)** cf Rendu3 section 2

t42: le robot spatial decide de créer un nouveau robot neutraliseur quand nb_update est un multiple de modulo_update = 100. Une seule particule ; un seul robot en reserve ⇨ 0.5pt	

t43: le robot spatial decide de créer un nouveau robot neutraliseur quand nb_update est un multiple de modulo_update ; context avec 4 particules et 3 robots en reserve. 3 robots neutraliseurs doivent être créés, de 3 types. Images pour 0 , 100, 200, 300, 365, 553 ⇨ 0.5pt	

Modifier constantes.h pour remettre le taux à 0.002 ; éditer seed(2) dans simulation

Column **t44** : **test perf avec desintégration (1pt)** cf Rendu3 section 2

Le programme ne doit pas planter ; on doit voir la creation de robots neutraliseur, la désintégration. Éventuellement une panne et destruction du robot puisqu'il n'y a pas de réparateurs (prog demo fin à 3442steps) Pénalité de -0.5pt si absence de test de collision au moment de la création	

Column **t45** : **test perf avec desintégration (1pt)** cf Rendu3 section 2

Le programme ne doit pas planter ; on doit voir la creation de robots, la désintégration. Idéalement un robot en panne avec un réparateur qui joue son role. (prog demo blocage à cause des collisions; pas de pénalité) Pénalité de -0.5pt si absence de test de collision au moment de la création	

ARCHITECTURE EVALUATION:

Nb points (max=0.5pts)	Module role / separation of functionalities
[A1] 0.5	Must use **argc** and **argv,** at least to check whether there is an argument.

[A2] Architecture features to check for the **Model sub-system**:

Nb points	Module role / separation of functionalities
[A2.1] 0.50	Simulation must declare a class ; simulation.h must NOT be included in the lower-level modules ;
[A2.2] 0.50	There must be NO dependency to GTKmm in any Model module
[A2.3] 0.50	The **robot** entities must be managed with a **hierarchy of classes** ; they can be defined in the same module or different modules

[A3] Architecture features to check for module **shape**:

Nb points	Module role / separation of functionalities (same as rendu1)
[A3] 0.50	The module **shape** has to be independent from higher level modules, including gui, and from GTKmm ; only the include of **graphic.h** is allowed Ex : including the appendix A = « constantes.h » in the shape module is a clear violation of the architecture specification.

[A4] Architecture features to check for module **gui**:

Nb points	Module role / separation of functionalities
[A4] 0.25	connection with the **Model** sub-system with **simulation.h only** but simulation.h can include other interfaces for its own class needs. OK to include shape.h and gtkmm.h

[A5] Architecture features to check for module **graphic** :
Check the report if this module is not present ; in such a case the gui module gather all the relevant information from the Model to manage the display with GTKmm.

ARCHI pt	If the module graphic is present: Module role / separation of functionalities
[A5] 0.5pt	Same rule as for [A3]: no dependency to higher level of the Model or to gui

The spreadsheet column shows the **default maximum of 2 point** for ARCHITECTURE.
=> **Remove the number of point indicated for each feature that is not achieved, but not more than 2 pts**.
In the spreadsheets column AH architecture violation comment, note down the corresponding **code(s) : e.g. [A1], [A2.1], [A2.3], [A3]** etc

4. CLASS ENCAPSULATION / MODULARIZATION: same as rendu1&2

[C0] Incomplete implementation: the max number of points is reduced in case of partial implementation. Do not waste time to figure out this in detail ; it should be obvious that a large fraction of the code is missing : ***Report the case 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 (indicate a warning if there are too many of them). Indicate a 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 AI shows the **default maximum of 2 points** for **CLASS**.

=> **Remove 1 point per public attribute or global variable** (max 2pt).

=> **Remove 1 point per interface that is not correctly externalized** (max 2 pt).

The total of removed points from C1 and C2 is maximum 2 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**.

5. **CODING STYLE: less criteria as for Rendu2 to spare time for execution tests**

[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).

Note: it is OK that "**case**" is not indented in the **switch** block but controlled instructions have to be indented.

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

[P2] Apart from two function/method of max 80 lines, all function/method size must not exceed 40 lines (+tolerance of 2 lines) with geany (with the default font size). Recommend to apply the principle of abstraction in case of too long function/method.

The spreadsheet column shows the default maximum of 5 points for STYLE

=> **remove 1 point max for [L1]**
=> remove 1 point max for [L2]
=> remove 1 point per function/method that is too long [P2]

In the spreadsheet column 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 AM.

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