CONCEPTION

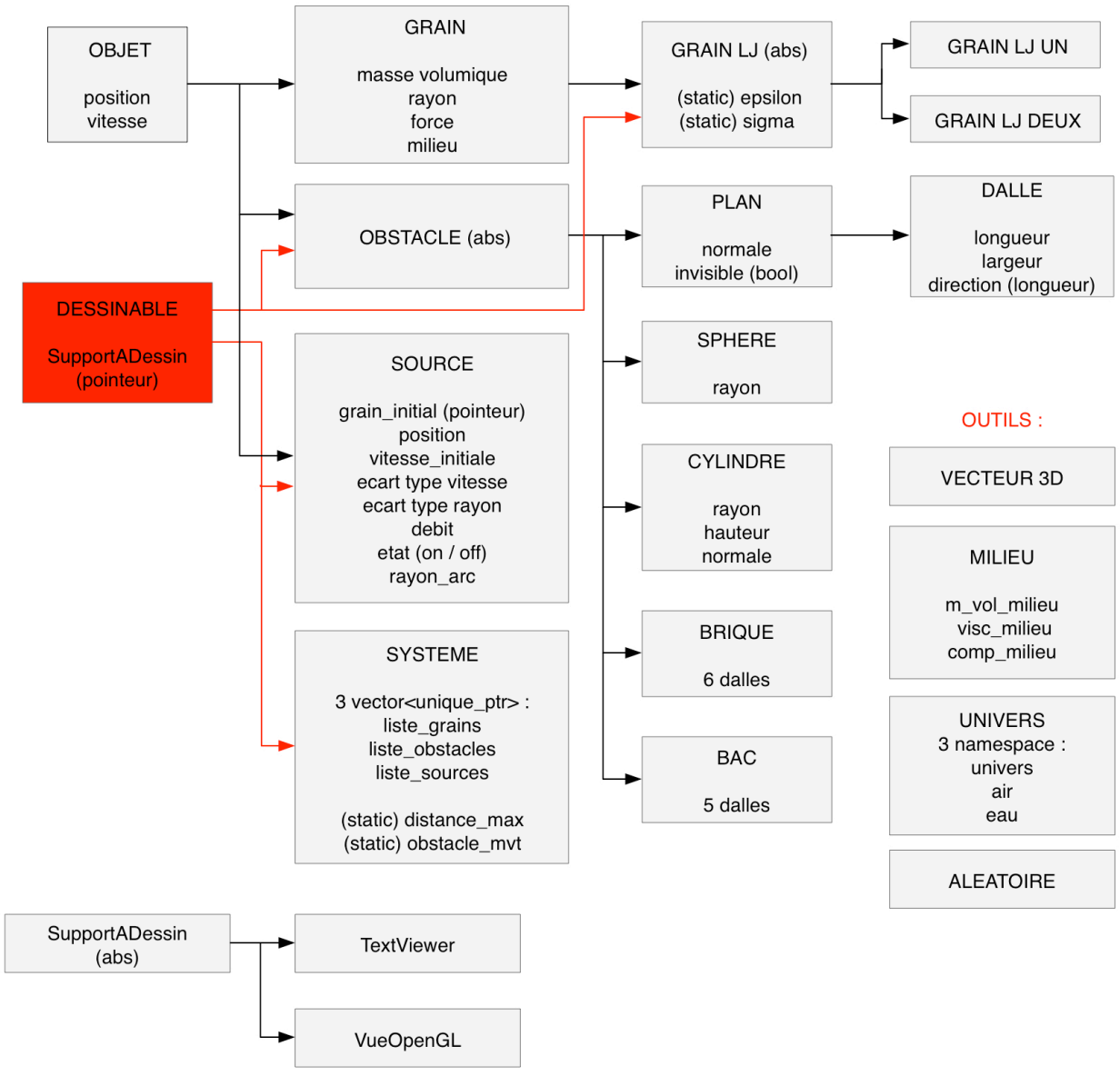
I. Introduction.
II. Liste classes, attributs et relations.
III. Critique et amélioration possibles du projet.

I. Introduction:

Toutes les données physiques sont en unités internationales. C'est a dire :
distance en (m), temps en (s), masse en (Kg). L'introduction de ces données peut être en unités différentes mais sont converties automatiquement en SI. En particulier, pour créer des objets, les distances sont en (mm) et les masses en (mg).

II. Liste Classes, Attributs et Relations:

Il y a différentes méthodes dans chaque classe qui sont expliquées soit en commentaire dans le code, soit dans le fichier Réponses.



(abs) = classe abstraite.

Les classes/namespaces « outils » sont utilisés dans la simulation pour l'affichage et le calcul.
- « SupportADessin » est nécessaire pour le « double dispatch » pour pouvoir dessiner le bon Objet de la bonne façon (méthode dessine polymorphique) et dans le bon Support (OpenGL ou Text).
- « Univers » contient les constantes physiques et « Milieu » prend les bonnes constantes physiques de « Univers » ce qui permet de faire un changement de milieu si on le souhaite.
- « Vecteur3D » l'outil le plus magique.

- **LISTE AMÉLIORATIONS PAR RAPPORT AU MINIMUM DEMANDÉ :**
  - class Objet : une position et une vitesse qui sont converties en m.
  - class Grain : ajout de l'attribut « milieu » , chaque grain se trouve dans un milieu (eau ou air dans notre simulation mais on peut rajouter un nouveau namespace dans « univers » et modifié constructeur dans « Milieu »)
  - GrainLJTUn et GrainLJTDeux ont deux couleurs différentes en mode graphique.
  - class Plan: ajout de l'attribut « invisible » pour pouvoir dessiner par exemple une vitre ou fermer la planche de Galton dans notre cas.
  - class Bac: Bac a eau pour montrer qu'on peut changer de milieu dans une partie de l'espace.
  - class Obstacle: ajout des méthodes : translation (translate dans la direction de la vitesse), rotate (rotation qui depend du type d'obstacle), bouger et changer_univers (utilisé certaines fois pour modifier les grains aux alentours).
  - class Source: ajout de l'attribut « rayon_arc » pour créer des grains sur un arc aléatoirement (moins de grains trop proches)

III. Critique et amélioration possibles du projet:

- Le plus gros problème avec la simulation c'est le fait que les grains traversent des obstacles. En effet, lorsque un nombre très important de grains sont cote a cote et sur un plan (par exemple) le fait qu'on calcul la force dans le meme ordre (dans système) fait qu'on bouge le grain qui est sur le plan en premier pour l'éloigner d'un autre grain et forcement il va traverser le plan. UNE SOLUTION : faire bouger le grain le plus haut par rapport au plan en premier pour laisser de l'espace aux autres grains pour s'éloigner du plan. Ceci est vrai pour tout obstacle. On a essayé d'effacer les grains qui essayent de traverser un obstacle , ca marche bien mais des fois la simulation crash donc on a enlevé cette tentative de correction.
- On n'as pas fais l'exerciceP12 et P13 faute de temps et fatigue, mais on a réfléchi sur la conception. L'idée est de partitioner l'espace en des cubes de coté « 2*sigma » tel que la partition contient la Sphere de rayon « distance_max » de la class « Système » . Chaque cube aura un tableau de pointeurs sur des « unique_ptr <GrainLJ » ou les indices des grains dans la « liste_grains ». Au lieu de calculer la force subis par les tous les autres grains , on calcul uniquement les cubes qui sont a coté (on gagnera en temps de calcul).