#Projet Programmation orientée objet (SSV)

##Réponses du groupe Hélène Zablit et Julie Korach

****

##Q1.1

Il faut coder clamp comme une fonction indépendante de la classe Collider

****

##Q1.2

Il faut coder la méthode à l'aide de 2 boucles imbriquées. La première qui modifie la largeur et la deuxième qui modifie la hauteur.

****

##Q1.3

Les objets potentiellement volumineux sont à passer par référence constante, donc les Collider.

****

##Q1.4

Les méthodes à déclarer comme const sur les fonctions qui ne modifient pas l'objet de la classe.

****

##Q1.5

Pour éviter de dupliquer du code, il faut écrire ces 3 fonctions en appelant d'autres fonctions comme isColliding, isColliderInside ou isPointInside.

****

##Q1.6

La surchage externe est nécessaire pour des opérateurs concernés par une classe mais pour lesquels la classe en question n'est pas l'opérande de gauche : c'est le cas pour le cout ou on souhaite surcharger la classe et non ostream donc on doit utiliser la surchage externe pour la dernière surchage (opérateur <<) mais pour les autres, il n'est donc pas nécessaire de faire une surchage externe, une surchage interne suffit.

****

##Q1.7

Les arguments à passer par référence constante sont les arguments volumineux qui ne sont pas modifiés comme des Vec2d ou des Collider.
****

##Q1.8

Il est judicieux de déclarer des méthodes comme const lorsqu'elles ne modifient pas l'état de l'objet, c'est le cas de beaucoup de méthodes que l'on a codé comme getradius, getposition, directionTo, distanceTo, isColliderInside, isColliding, isPointInside...

****

##Q2.1

La taille de cells_ est
`getAppConfig().world_size*getAppConfig().world_size`

****

##Q2.2

La séquence d'étiquettes ["simulation"]["world"][default cells"] du fichier .json correspond à `getAppConfig().world_cells` et la séquence d'étiquettes ["simulation"]["world"][default size"] du fichier .json correspond à
`getAppConfig().world_size`

****

##Q2.3

Il faut appeler la méthode reloaConfig, puis reloadCacheStructure et enfin updateCache dans reset. L'ordre est important parce que nbCells_, cellSize_ et cells_sont initialisés dans reloadConfig et sont utilisés dans reloadStructure et cette fonction initialise les vertexes et renderingCache qui sont utilisés dans updateCache.

****

##Q2.4
Les méthodes sont un avantage car elles permettent de sélectionner le dossier sans quoi il faudrait donner un nom précis de fichier et donc si le fichier changeait de nom le code ne serait plus valable.

****

##Q2.5
Pour mettre à jour les attributs nécessaires au rendu graphique du terrain après l'initialisation de cells_ depuis un fichier, la méthode loadFromFile doit appeler les méthodes reloadCacheStructure et updateCache pour traduire chaque cellule de cells_ en vertex et pour permettre ensuite la génération du terrain.

****

##Q2.6

On a choisi de représenter l'ensemble seeds_ en un vector de Seed initialisé à une taille fixe (nbGrassSeeds_+nbWaterSeeds_) mais peut changer d'une exécution à l'autre puisqu'il s'agit d'un vector.

****

##Q3.1

Pour l'instant notre Env se compose uniquement d'un monde donc le corps de Env::reset se compose seulement d'un appel à la fonction reset de World. Idem pour Env::drawOn. Pour Env::update, il se compose pour l'instant uniquement d'un corps vide.

****

##Q3.2

Pour coder la méthode Env::loadWorldFromFile il suffit d'appeler la méthode loadFromFile de world. Cette méthode s'utilise dans le constructeur de Env.Le fichier chargé sera getApp().getResPath() + getAppConfig().world_init_file.

****

##Q3.3

L'algorithme d'humidité doit être appelé à la fin de loadFromFile pour calculer l'humidité du terrain qui vient d'être loadé, mais également dans reset pour calculer l'humidité du nouveau terrain si on décide d'en charger un nouveau avec la touche r.

****

##Q3.4

Il faut mettre la couleur de l'humidité sur chaque cellule en même temps que les transparences.

****

##Q3.5

Les fleurs peuvent être modélisées comme des colliders, de ce fait, la classe Flower héritera de Collider. Il y a un lien est-un.

****

##Q3.6
Si l'on tire au sort l'indice dans la méthode de dessin, la texture de la fleur va changer à chaque appel à la méthode de dessin. Pour assurer que le choix de la texture se fasse une fois pour toute lors de la création de la fleur, on peut mettre un attribut texture à la fleur et l'initialiser dans le constructeur de la fleur.

****

##Q3.7
Si l'on anticipe que plus tard, il peut y avoir différentes sortes de fleurs (par exemple des fleurs contenant des toxines pour les abeilles), on peut coder cette collection comme un vector de pointeurs de fleurs.

****

##Q3.8
Les fleurs appartiennent à l'environnement (Env) et ne peuvent vivre en dehors de lui. Donc, dans le destructeur de Env, on ne doit pas détruire que le terrain, on doit également parcourir le vector de flower et détruire toutes les fleurs, puis clear le vector.

****

##Q3.9
La régénération d'un environnement (reset ou le chargement du terrain depuis une fichier) implique de repartir d'un nouveau terrain et donc de supprimer aussi les fleurs. Dans reset, on doit donc aussi clear le vector de flower*

****

##Q3.10
Dans Env::draw0n, on doit parcourir le vector de flower, et appeler la méthode drawn de flower pour chaque fleur.

****

##Q3.11

Dans update de Env, il faut itérer sur le tableau de fleur et appeler update() pour chaque fleur

****

##Q3.12
Il faut donc modifier dans Env la méthode update. Il faut parcourir le vector de flower* et pour chaque fleur, il faut appeler la méthode update de flower.

****

##Q3.13
Pour faire en sorte que les fleurs de votre simulation meurent/disparaissent si leur quantité de pollen est/devient nulle, il faut modifier Env::update. Il faut suprimer le pointeur de la fleur et le mettre à zero si la quantité de pollen de la fleur est négatif et tout à la fin, il faut supprimer tous les pointeurs nuls dans flowers_.

****

##Q3.14
Il serait bon de faire hériter de Drawable et Updatable la classe Flower(puisqu'elle a une méthode draw0n et update) et de faire hériter de Drawable la classe World(puisqu'elle a seulement une méthode draw0n). Cela leur permettra d'hériter directement des méthodes publiques de Drawable et/ou Updatable et donc cela sera plus simple à concevoir.

****

##Q3.15
On peut mettre en attribut un pointeur de world dans Env au lieu de mettre un attribut World.

****

##Q3.16
Dans Env, la méthode update doit appeler au début la méthode update de FlowerGenerator afin de permettre au générateur de générer des fleurs dans l'environnement.

****

##Q4.1

Il est préférable de modéliser l'ensemble des abeilles d'une rcuhe comme un vector<Bee*> car ce sont de gros objets à manipuler. C'est plus efficace de travailler avec des adresses.

##Q4.2

La classe Hive est un collider, est drawable, et est updatable, elle va donc hériter de ces 3 superclasses. Elle va prendre en attribut un double pour représenter sa quantité de pollen, ainsi qu'un vecteur de pointeur sur les abeilles qui sont rattachées à cette ruche.

****

##Q4.5

Une ruche appartient à un environnement donc si l'environnement est détruit les ruches aussi.

****

##Q4.7

Il fait modifier les méthodes reset et drawOn de Env. Pour la régénération d'un Env, il faut régénérer les ruches donc clear le tableau dans reset de Env. Pour le dessin d'un Env, il faut pouvoir dessiner les ruches donc il faut appeler drawOn de Hive dans drawOn de Env.