JOURNAL DU PROJET DE PROGRAMMATION

Au vu de la situation particulière nous travaillons sur nos propres ordinateurs à la maison. Cela implique quelques petites conséquences: sur un ordinateur la touche V ne marche pas systématiquement (toute absence ou doublement de cette lettre est dû à ça). Pour l'autre membre du groupe c'est la connection internet qui est défectueuse par moment.

SEMAINE 4 :

Nous avons codé la classe *CircularBody* avec son constructeur, son constructeur de copies, ses méthodes et ses opérateurs spécifiques.

Cette classe permet de représenter des corps circulaires (cercles) et de les situer dans l'espace grâce à la position du centre et la taille du rayon. Grâce aux méthodes codées nous pouvons déplacer ces corps, les comparer à d'autres (ils se touchent ? ils s'englobent ?) ou juste changer la taille du rayon, ou la position du centre.

Nous avons travaillé ensemble durant les heures du TP.

SEMAINE 5 :

Nous avons codé les classes : PetriDish, Nutriment et Lab.

La classe **PetriDish** hérite de CircularBody. Elle possède donc les attributs d'un CircularBody, et en plus deux tableaux (un pour les nutriments et un pour les bactéries) et un attribut température. Cette classe possède plusieurs méthodes, notamment:

- un constructeur et un destructeur qui doit désallouer la mémoire utilisée pour ses nutriments et ses bactéries (qui sont des pointeurs);
- une méthode d'évolution dans le temps;
- des getter et setter;
- une méthode de dessin pour le contour de la boite, les nutriments et les bactéries (reste à coder);

Étant donné que la PetriDish est un objet volumineux et que nous ne devons pas la copier, nous avons aussi supprimé le constructeur de copie et l'opérateur =.

La classe **Nutriments** hérite de CircularBody et a comme attribut spécifique une quantité. Pour ses méthodes elle a :

- un constructeur qui permet de mettre en relation taille et quantité;
- setter pour la quantité;
- une qui permet de faire des traitement sur les prélèvement de quantité;
- une méthode de dessin, pour laquelle nous avons dû faire un peu de “bricolage”: la quantité de nutriment n'atteint pas 100 (ce n’est pas à cause d’un opérateur de comparaison </<=), nous avons donc résolu le problème en ajoutant 0,5 à la quantité;
- une méthode d’évolution dans le temps (qui a été difficile de coder);
- un raccourci

Nous avons oublié le destructeur pour le moment.

La classe **Lab** est caractérisée par une PetriDish. Possède comme méthodes :

- un constructeur qui place la PetriDish au centre de la fenêtre
- une méthode pour tester si un CircularBody est à l’intérieur
- une méthode de dessin et évolution dans le temps
- une méthode d’ajout de nutriment
- des méthodes de traitement de la température
- une méthode de réinitialisation

Pour cette classe nous avons oublié de coder le destructeur et d'éliminer l’opérateur d’affectation.

Les tests ont été effectués: la majorité des fonctionnalités est opérationnelle. Il reste une partie oubliée pour les fonctionnalités de la touche C (réinitialisation de la température).

Pour la répartition du travail: chacune de nous a pris en charge une des deux parties. On a commencé à traiter les questions les plus simples chacune de notre côté. Puis pour les questions qui nécessitaient plus de réflexion et pour la deuxième partie qui comporte des dépendances entre les classes on l’a fait ensemble via zoom. Enfin on a répondu aux questions de notre partie sur un fichier commun.

SEMAINE 6:

Cette semaine a été moins productive que la précédente par manque de temps principalement. Nous avons réparti le travail de la manière suivante : il y a 4 sous-étapes pour cette étape et nous avons commencé avec les deux premières (chacune une).

Pour la partie sur les **nutriments différenciés et générés automatiquement** la hiérarchie des nutriments a été faite.

La partie **MutableNumber** a été assez facile à coder, malgré des incompréhensions sur le fonctionnement de la fonction fournie bernoulli.

Par contre la partie **MutableColor** a été plus compliquée, surtout pour le constructeur dans lequel on devait initialiser un tableau d'objets.

Nous n'avons pas encore eu le temps de regrouper notre code et de faire les tests, mais ce sera fait ce week-end.

SEMAINE 7:

Nous avons fini de coder ensemble et tester les parties 3.1, 3.2 en début de semaine: pas de problème notable même si la partie 3.1 nous a demandé beaucoup plus de temps que prévu pour ce qui concerne la fonction update de **NutrimentGenerator**.

La partie 3.3 après avoir fini de tout coder la classe Bacterium n'était pas reconnue à cause d'un problème d'inclusion. Une fois le problème résolu il a fallu résoudre un problème de crash : à chaque lancement du programme après avoir créé une bactérie le programme s'arrêtait après quelques secondes. Il s'avère que le problème venait d'un oubli de return dans une méthode. On a revu aussi l'élimination des nutriments et bactéries qui n'était pas au point. Il reste quand même un problème au niveau du temps d'attente entre deux repas d'une bactérie : elle attend le double du temps d'attente rentré dans le fichier Json.

SEMAINE 8:

Cette semaine nous avons commencé avec la partie 3.4, que nous avons codée ensemble malgré quelques problèmes de connexion internet et résolu le problème d'attente entre deux repas.

Nous avons trouvé l'énoncé expliquant la méthode move peu clair: il y avait des aspects que nous avons compris seulement après avoir essayé de compiler sans succès le SimpleBactTest.

Pour la partie de la direction aléatoire on a choisi de générer 20 directions aléatoires et de prendre celle qui possède le score le plus avantageu. Nous avons jugé que c'était plus intéressant pour la simulation même si ce fut plus laborieux. Nous avons donc créé une méthode Vec2d best_direction dans Bacterium qui crée 20 struct ayant un Vec2d direction et un double score. Nous générons aléatoirement des directions et nous calculons le score associé. Ensuite nous itérons à la recherche du meilleur score que nous retournons.

Pour la partie concernant la flagelle, nous avons décidé de rendre la méthode drawOn de Bacterium virtuelle. Ainsi nous avons codé drawOn de SimpleBacterium en appelant d'abord

drawOn de Bacterium (ceci nous permet de dessiner le “cercle” de la bactérie) et en ajoutant la partie codant pour le dessin de la flagelle.

Pour la méthode division le code n’a pas posé trop de problème cependant on remarque qu'étant donné le changement de direction influencé par la quantité de nutriments quand une bactérie se divise elle ne prend pas forcément la direction opposée car elle est très vite déviée. Nous appelons cette méthode dans PetriDish::update() car cela nous semblait plus simple: il suffit d’ajouter les bactéries issues de la division dans le vector de nouvelles bactéries, puis _>ajouter les nouvelles bactéries aux vieilles et ensuite éliminer tous les nullptr (qui peuvent venir de la division ou de la mort d’une bactérie). Nous trouvons cela mieux que d'appeler division dans Bacterium::update car il aurait fallu ajouter une méthode à la classe PetriDish pour pouvoir ajouter des bactéries au vector de nouvelles bactéries.

Par contre la partie mutation a posé des problèmes … Nous avons eu un problème avec l’initialisation d’un MutableNumber. Ceci a été résolu en ajoutant le constructeur par défaut pour éviter d’avoir des MutableNumber vides. Selon les assistants on aurait aussi pu utiliser la méthode insert() de std::map pour résoudre notre problème, ceci aurait été la meilleure solution, mais la plus difficile à implémenter. Nous avons donc décidé de choisir la voie la plus simple, car c’est celle que nous comprenons mieux afin d’éviter d’avoir un code que nous mêmes nous ne comprenons pas très bien.

Par contre pour l’instant nos bactéries ne mutent pas leur couleur.

VACANCES DE PÂQUES:

Maintenant nos SimpleBacterium changent de couleur, nous avions oublié le & dans l'itération for(auto**&** …) sur les éléments d’un MutableColor dans sa méthode mutate().

Nous avons divisé en deux la partie 4: chacune a pris un type de bactérie à coder.

Pour les bactéries à tentacules tout s’est bien déroulé, mais cela n’a pas été facile. Il reste encore un problème car l’algorithme pour trouver la meilleure direction ne semble pas toujours fonctionner: certaines bactéries se dirigent bien vers les sources de nutriments avec le plus grand score, mais d’autres semblent choisir la direction entre deux sources de nutriments, ce qui est un problème.

Pour les bactéries swarm nous avons codé les grandes lignes mais de nombreux problèmes persistent. Notamment au niveau de la force (et donc de la méthode move) nous avons une erreur d'architecture. On suppose que cela vient des différents liens d’héritage et des liens entre Swarm, SwarmBacterium et SwarmForce (en effet nous avons choisi de créer une classe SwarmForce dans Utility qui hérite de DiffEqFonction comme nous l’avions fait pour NullForce de SimpleBacterium).

SEMAINE 9:

Nous continuons à travailler sur la partie des SwarmBacterium. Le test 15 a marché dans les grandes lignes, mais nous devons améliorer des choses. Premièrement l'initialisation du Leader. Dans le constructeur de SwarmBacterium nous disons que si la bactérie que nous ajoutons est la première du groupe (swarm) alors c'est le leader. Mais cela semble être un effet de bord. Ainsi nous avons modifié l'initialisation du leader dans update de Swarm afin que si le swarm possède qu'une seule bactérie elle soit automatiquement le leader. Si le swarm possède 2 ou plus bactéries il cherche le meilleur leader.

SEMAINE 10 :

Nous avons profité de la séance de TP pour résoudre les problèmes majeurs de SwarmBacterium : problème notamment avec la force et quelques réglages avec leur déplacement. Ainsi pour ce qui est de la force, le problème d'architecture qui nous bloquait venait du fait qu'on ait mis une classe SwarmForce dans Utility alors que le Cmake ne remplit pas cette fonction. Nous avons donc déplacé notre force dans le Lab, ce qui est cohérent puisque c'est une partie du lab et non plus juste un outil.

Ayant reçu le compte rendu de notre assistant nous avons pu améliorer un peu notre code : changements de noms, modularisation de la méthode move. Certains points restent à éclaircir avec lui concernant l'utilisation de final, des erreurs de comportement des TwitchingBacterium et de l'organisation entre private et protected.

Nous nous sommes réparties la partie 5 pour la semaine à suivre.

SEMAINE 11:

Pour la partie des statistiques, le premier test s'est bien passé, sans problèmes majeurs. Ce qui a demandé le plus de concentration c'était la réflexion sur comment modéliser l'ensemble des graphes avec les libellés et leur identifiant. Pour faire cela nous avons décidé de faire une map avec un unsigned int (qui représente l'identifiant) et une struct Graphe-Libelle qui contient donc un unique_ptr sur un graphe et un string pour le nom. En faisant cela on peut garder le lien entre les trois éléments: en ayant un élément on peut trouver les deux autres dans la même structure de données. Pour pouvoir compter les différents type de bactéries on a pris la décision de créer une map avec comme identificateur un string (le type de la bactérie), et comme valeur un entier positif(le nombre). Grâce à des méthodes d'incrémentation et de

décrémentation dans la Petridish et le Lab ou a pu, dans les constructeurs, destructeurs et méthodes clone de chaque type, gérer leur nombre respectif dans la map.

Pour la partie différenciation de nutriments ce fut assez facile à coder car la partie était très guidée, mais les questions nous ont demandé un peu plus de réflexion. Le test associé est passé sans problème. Cette partie nous a aussi permis de corriger une petite erreur dans la méthode takeQuantity() de nutriment.

Au moment du test final nous nous sommes rendues compte que de nombreuses erreurs apparaissent. Certaines venaient d'une erreur dans la méthode move des bactéries simples et à comportement de groupe ce qui fut facil à corriger (on ne sauvegardait nulle part l'ancienne position pour le calcul de l'énergie dépensée donc nos bactéries ne perdaient plus d'énergie en se déplaçant) nous avons soulevé la question d'une modulation supplémentaire de move() que nous réaliserons si nous avons le temps à la fin. Mais d'autres erreurs persistaient encore autour des SwarmBacterium après de nombreux tests et de nombreux contrôles sur les tests précédents nous nous sommes rendues compte qu'elles apparaissent déjà au niveau du test de l'étape 4 cependant il faut attendre un bon moment et ajouter beaucoup de bactéries pour les voir. On suppose que cela vient d'une mauvaise gestion de mémoire liée aux pointeurs abondants dans cette partie.

Effectivement notre code comporte plusieurs erreurs au niveau des swarms notamment sur la méthode clone(), la mise à jour du leader dans swarm, et le destructeur du swarm.Nous avons donc modifié plusieures choses : notamment l'élection du leader, qui maintenant est indépendante de la quantité de bactéries du Swarm (du moment qu'il y en a au moins une), l'ajout des clones au Swarm, et une meilleure gestion de la mort des pointeurs qui doit uniquement être faite par la PetriDish (car c'est elle qui a la propriété). Ainsi après la correction de ces erreurs nous nous sommes appelées une deuxième fois cette semaine pour voir si tout allait bien lors de la compilation et de l'exécution des tests.

Nous ne voyons aucune erreur majeure de comportement dans nos bactéries, à part certains TwitchingBacterium qui, quand nous laissons longtemps la simulation, ne bougent plus. Nous pensons que c'est dû au fait qu'ils mutent beaucoup et donc la longueur et la vitesse de leur tentacule sont proches de 0. Nous voulons donc modifier le code pour éviter d'avoir des clones avec ces caractéristiques. Pour faire ceci nous avons modifié la méthode mutate en évitant tout paramètre égal à 0 pour toutes les caractéristiques.

SEMAINE 12:

Nous avons finalisé notre code en l'épurant et le rendant plus lisible. Nous nous sommes ensuite penchées sur les extensions et avons pris la décision d'ajouter un type de bactérie : FriendlyBacterium, qui distribue des Help (deux nouvelles classes s'ajoutent à notre code).

Les FriendlyBacterium fonctionnent dans les grandes lignes comme les SimpleBacterium. Elles lâchent tous les dt des Help qui viennent booster l'énergie des autres bactéries (les Simple et les Swarm). Les Help ne sont pas consommables et disparaissent au bout d'un certain temps. Nous avons fait ce choix pour avantager les SimpleBacterium et les SwarmBacterium par rapport au TwitchingBacterium car elles sont beaucoup moins résistantes.

Nous avons aussi introduit des obstacles qui sont simplement des lignes qu'on peut modifier sur le fichier json de façon à ce qu'elles soient horizontales ou verticales. On les ajoute avec la touche 'o' du clavier et on peut supprimer le dernier ajouté avec la touche 'effacer'. Nous aurions pu créer une superclasse Obstacle et des sous classes Ligne, Cercle … mais comme nous avons décidé de ne faire qu'une sorte d'obstacles (étant donné qu'on devait déjà finir les FrendlyBacterium) nous avons juste créé une classe Obstacle. Les simpleBacterium rebondissent sur les obstacles et prennent la direction opposée à leur direction initiale pendant 2s (à aviser encore) de façon à ce qu'elles ne restent pas bloquées face à l'obstacle dans le cas où la source de nutriments avec le plus grand score serait de l'autre côté de l'obstacle. Cette solution n'est pas la meilleure : on a pensé à mettre un score très bas si la trajectoire comporte un obstacle mais par faute de temps nous avons laissé la dernière. Nous aviserons peut être la semaine prochaine. Les swarmBacterium réagissent de la même façon que les simples et les twitching ont l'incapacité de traverser l'obstacle avec leur grip qui se rétracte au contact d'un obstacle. Un problème advient quand la source de nutriment et l'obstacle sont au même endroit le grip attire la bactérie qui déborde sur l'obstacle et peut donc envoyer son grip pour la prochaine fois de l'autre côté. Il faudra aussi qu'on modifie cela la semaine prochaine.

SEMAINE 13:

Cette semaine nous avons finalisé notre projet, nous ne reviendrons pas sur le code avant le rendu final. Nous avons donc une partie 6 qui comporte les extensions sur les FriendlyBacterium et une partie7 qui comporte la partie sur les obstacles. Nous avons passé beaucoup de temps sur les obstacles : nous avons fini par créer une superClasse RectangleBody sur le modèle de CircularBody qui comporte des méthodes utiles comme isColliding et contains. Si le code compile il y a encore beaucoup de problèmes avec le comportement des bactéries face aux obstacles et quelque défaillance: on peut créer des obstacles en dehors de la PetriDish (la partie du code nous l'empêchant nous affiche une erreur d'architecture) et une anomalie advient parfois au bout d'un certains temps : les bactéries restent bloquées devant l'obstacle ou traversent. De plus les nutriments peuvent se créer sous les obstacles. De plus une erreur de conception réside peut être dans notre code : nous avons pris la décision que les bactéries qui rencontrent un obstacle prennent la direction de déplacement opposée à celle de leur arrivée pendant 2s sans être influencées par les nutriments qui les entourent afin qu'elles ne restent pas bloquées si la meilleure source de

nutriment est derrière l'obstacle. Cependant ce choix de conception fait que si la bactérie est entre deux obstacles et qu'elle a une direction perpendiculaire elle reste bloquée en faisant des aller-retours. De même pour les tentacules des TwitchingBacterium. Ainsi à cause de ces trop nombreuses erreurs et du temps qui nous manque avant le rendu final, nous avons décidé que nous ne présenterons que la partie 6 et donc supprimerons les obstacles de notre projet.

Nous avons écrit le fichier README, relu nos réponses et notre journal pour le rendu final.