Groupe 25: Léa Pistorius, Méline Cretegny

JOURNAL

Semaine 4 (11/03 - 17/03)

- Mise en place du répertoire de travail
    - Création des différents google docs
    - Création du projet1 et mise en place de geany
- Méthodes générales
    - Il nous a fallu un moment pour comprendre comment programmer le monde torique, et nous nous sommes même corrigées en y revenant
    - La classe CircularCollider n'a pas été trop compliquée à réaliser
- Gestion du déplacement
    - La méthode directionTo a été de loin la plus compliquée cette semaine, demandant d'abord une compréhension de l'objectif de celle-ci et de son utilisation de la fonction clamping. Une fois les concepts compris, il fallait optimiser la méthode pour éviter des calculs fastidieux et répétitifs, d'où le tableau et la méthode checkShortestDistance.
- Gestion des collisions
    - Cette partie était plutôt claire et rapide. La seule éventuelle difficulté était de savoir quelles paramètres passer et comment.
- Organisation
    - Étant notre première semaine de projet, notre organisation et la répartition du travail n'étaient pas les plus efficaces. Nous avons programmé ensemble pour la plupart des méthodes ce qui nous as pris beaucoup de temps. Le point positif est que nous sommes les deux au clair du déroulement du projet et le comprenons mieux.

Semaine 5 (18/03-24/03)

Pendant les 3 heures de TP, nous avons réussis à tout finir l'Étape 1. Puisque nous avions du mal à comprendre les commandes des tests sur le terminal, nous avons transféré l'exécution de scons dans Geany afin de corriger nos erreurs de programmation. Des erreurs de frappe ont été les plus fréquentes permettant une correction plutôt rapide.

Nous avons donc passé à l'Etape 2 et nous avons essayé de mieux répartir le travail cette fois-ci pour que nous soyons plus efficaces. Léa a codé la Poursuite de cible et Meline sa fonction force ainsi que le Lieu de vie.

Semaine 6 (25/03-1/04)

Groupe 25: Léa Pistorius, Méline Cretegny

Nous avons réussi à tout finir le code de ChasingAutomaton et d'Environnement et avons aussi commencé le code de la classe Animale. Nous avons laissé les tests pour les heures de TP du mardi.

Pendant les 3 heures de TP nous avons seulement eu le temps de comprendre pourquoi notre EnvTest ne voulait pas fonctionner ni tous les autres tests. Au tout début nous avons utilisé les guillemets pour inclure des documents hpp, puisque les <> créait un problème de compilation. Nous avons compris qu'afin de compiler correctement nous ne pouvons seulement utiliser la commande dans Geany (sans scons) ou dans le terminal scons NomTest-run.

Ensuite, lorsque la classe Environment ne causait plus de problèmes nous sommes passées au ChasingAutomaton. Par contre, lorsque nous mettions une cible lors du test, le fantôme fuyait la cible au lieu d'être attiré par elle. Cette erreur fut rapidement corrigée dans la méthode forceAttraction.

Nous nous sommes arrêtées au test 6 de la troisième sous-partie. Celui-ci nous pris du temps à corriger, nous avons confondus les attributs "ViewAngle" de l'animal et la méthode getViewAngle retournant l'angle maximale (pareil pour la distance). Du coup, on se demande si on a vraiment besoin des deux…

Pendant cette semaine la distribution du travail a été mieux gérée. On relit chacune la partie de l'autre ce qui aide énormément pour debugger le problème survenu lors des tests.

Semaine 7 (2/04-7/04)

En TP, nous avons pu finir complètement la deuxième partie et se lancer sur la troisième. La méthode getTexture, utilisant le polymorphisme, a été la plus difficile a comprendre mais après relativement rapide à mettre en place.

Ce qui nous a le plus bloqué pendant la semaine (nous avons passé plus de 10 heures combinées) était le test 10. Nous avions toujours des petites erreurs d'inclusion de fichier. Nous n'avons pas réussi à résoudre seules un problème concernant la méthode draw donc nous avons dû attendre la séance pour demander à un assistant afin d'éviter d'y passer des heures supplémentaires sans progrès.

Semaine 8 (7/04-14/04)

Nous avons réussi à avancer jusqu'à la partie 3.3 cette semaine. Les énoncés de cette partie ne sont pas particulièrement dur à comprendre et à coder, par contre elles mènent a énormément de petits problèmes de compilation ou d'exécution, ce qui entraîne beaucoup d'heurs de correction de code.

Cela implique que nous n'avançons pas au même rythme qu'avant car le debugging prend la majorité de nos heurs de code. Malheureusement nous avions espérés avancer plus rapidement donc nous prenons du retard alors que nous passons beaucoup de temps sur ce projet.

Semaine 9(14/04-21/04)

Cette semaine a été plutôt efficace. Nous avons réussis à enchaîner les tests plutôt rapidement et nous nous sommes arrêtées au test 18 (nous continuerons pendant les vacances afin d'avoir fini l'étape 3 a la reprise des cours).

Nous nous sommes faites avoir avec les segmentation faults au début du test 18 puisque nous ne vérifions pas toujours que le tableau n'ait pas de nullptr. Pour les résoudre, nous avons fait recours aux dévermineur.

Ce qui a été le plus dur à coder et surtout à conceptualiser était l'état FEEDING impliquant la disparition des entités mangées ainsi que l'arrêt de l'animal puis sa reprise de vitesse. Pour son changement de vitesse (associé à l'attribut normeVitesse de l'Animal), nous avons fait qu'il devienne 0.01 lorsqu'il était en pause (puisqu'à 0 il ne démarrait plus). C'est quelque chose que nous voulons changer par la suite dans la partie "Perfectionnisme".

Toutes les parties antérieures au Test18 sont fonctionnelles.

Vacances

En visualisant ce que produisait notre programme, nous avons remarqué que les animaux mangent leurs cibles sans être vraiment dessus mais dès qu'ils les touchent. Nous avons compris notre faute: la méthode UdateState, apres avoir analyse l'environnement, testait si "un CircularCollider isColliding avec un autre" alors que nous voulons savoir si la cible est "dans" l'animal. Nous avons alors utilisé l'opérateur > qui vérifie si l'élément passé en paramètre (la cible) est "dans" l'animal. Notre utilisation de isColliding était aussi fausse car nous aurions dû utiliser l'opérateur | qui utilise la méthode isColliding qui est en privé (alors que nous l'avons déplacée en publique ce qui n'était pas très pertinent).

Définition de la méthode virtuelle pure OrganicEntity::update qu incrémente simplement le tempsEnVie d'un OrganicEntity.

Groupe 25: Léa Pistorius, Méline Cretegny

Semaine 10 (29/04-05/05)

Nous avons réussi à résoudre nos problèmes par rapport aux gerbilles et aux scorpions qui ne pouvaient pas s'afficher. C'était un problème que normeVitesse n'était pas initialisée dans le constructeur d'Animal et qu'ensuite quand une gerbille ou un scorpion se créait, sa perte d'énergie (qui était en fonction de la normeVitesse) enlevait trop d'énergie ce qui faisait mourir les animaux directement.

Nous avons continué le test de reproduction (20) et test de fuite (21), tout les deux réussis. Par contre dans mating, cela arrive que 2 scorpions mate avec un autre, ce qui entraîne un threesome, ce qui apparemment n'est pas un problème.

Un des problèmes assez récurrent etait également que quand un scorpion voyait une gerbil, donc allait dans l'etat food_in_sight. Cette gerbil sortait ensuite de son champ de vision et le scorpion gardait la gerbil en tant que cible, donc la suivait. Nous avons réussi à résoudre le problème en initialisant cible et mate à nullptr dans la méthode analyzeEnvironnement().

Pour la fuite, nous avons résolu le problème de la condition pour que la gerbil soit en état RUNNING_AWAY en mettant une constante dans le fichier json (et Config): distMaxEnemy. Cette constante que l'on a mis à 450 permet que seulement à partir d'une certaine distance la gerbille se met ou s'enlève de cet état. Cette condition est mise dans la méthode updateState() et signifie que la gerbil passe en état WANDERING une fois qu'elle est a plus de cette distance du scorpion. Nous avons choisi la constante 450 parce que c'était la même taille que la distance de son champ de vision.

Nous commençons l'étape 4 juste à la fin de la semaine. En ce qui concerne la classe Wave, nous avons décidé d'utiliser l'attribut position de CircularCollider pour y stocker le point de départ de l'onde. Pour le rayon, nous avons créé un attribut rayonInitial dans Wave et utilisons l'attribut rayon de CircularCollider pour les mises à jour de celui-ci. Nous avons aussi ajouté les attributs energie et intensity afin qu'ils puissent être mis à jour facilement avec la méthode update() dans Wave.

Semaine 11 (06/05-12/05)

Tout au début de la semaine, nous avons corrigé nos erreurs de conception et de codage qui nous étaient indiquées dans le feedback du rendu intermédiaire dans l'étape 3 et donc la 4 également. Les étapes précédentes ne sont alors pas autant rigoureuses.

En commençant à compiler notre début de partie 4, nous avions des problèmes avec la mise à jour des fichier json, puisque nous avions ajouté des constantes qui ont été effacées lors du transfert de la partie 3 à la partie 4.

Nous avons pu coder la class Wave et la class Rock, tout en changeant la class Environment. Dans Environment, nous avons ajouté une list<Wave*> afin de pouvoir coder la méthode addWave(), qui fait un push_back dans la liste et une list<CircularCollider*> pour les obstacles.

Afin de pouvoir faire évoluer l'Environment pour les waves et les rocks(obstacles), nous avons fait des itérations qui font appel aux propres méthodes update de chacune des classes. Nous avons également modifié la méthode clean() pour qu'elle efface les ondes et les obstacles quand nous touchons sur la touche R (qui reset l'environnement).

Pour la conception d'une onde circulaire, nous avons du rajouter /DEG_TO_RAD afin que l'onde aille de -PI a PI en radian au lieu de degrés. Un typedef Arcs, qui est un std::pair<double, double>, nous a facilité le codage de la classe Wave pour pouvoir fragmenter l'onde en plusieurs arcs.

Nous sommes arrêtés au test 23.

Semaine 12 (13/05-19/05)

Le reste de la partie 4.1 ne nous a pas prit trop de temps, a part de comprendre que les ondes devaient etre traitees similairement aux Food gérés par FoodGenerator et qu'elles n'appartiennent pas aux gerbils mêmes mais sont utilisées par celles-ci. La partie 4.2 a été plus difficile niveau conception. Il nous a fallu un moment pour comprendre comment gérer l'interaction entre un Sensor et un NeuronalScorpion (que le scorpion a des senseurs, tous avec un index, et que eux "appartiennent" à un scorpion donc possède un pointeur sur lui).

Pour la classe Sensor, dans la méthode updateInhibitor(), nous avons mis en place des conditions afin de mettre inhibitor au minimum de l'intervalle (0) s'il est plus petit que 0, ou au maximum de l'intervalle s'il est au dessus de son maximum. Cela permet de garder inhibitor dans le bon interval.

Pour l'activation des sensors seulement pendant un temps limité, nous avons choisi de faire un compteur dans la class Sensor, initialisée à sf::Time::Zero et commence que lorsque le sensor est activé. Dans NeuronalScorpion, nous avons alors créé deux autres méthode updateCompteur(index,dt), qui ajoute dt au compteur à chaque pas de temps, et verifyCompteur(), qui compare la valeur du compteur actuel une constante. Si la valeur du compteur est plus élevée, NeuronalScorpion va réinitialiser tous ses senseurs avec la méthode reInitialisation dans Sensor qui met score et inhibitor à 0, et state à PAS_ACTIF.

Groupe 25: Léa Pistorius, Méline Cretegny

Pour le calcul du score du NeuronalScorpion, nous faisons un appel à la méthode updateScore() dans update() puis nous l'évaluons ensuite dans updateState() afin de vérifier si le scorpion passe à l'état MOVING.

Lors de la compilation, nous avons réalisé que le NeuronalScorpion ne peut pas simplement modifier les attributs privés de l'Animal. Nous avons alors codé plusieurs getters/setters pour palier à ce fait.

Semaine 13 (20/05 - 27/05)

Lors du TP, essayé de comprendre où étaient nos erreurs de la partie 4.2. Notre code compile mais ne fonctionne pas exactement comme nous le voulons. Lorsque le scorpion reçoit des ondes, étant dans l'état MOVING, il ne se dirige pas vers le centre de l'onde mais a l'air de suivre l'arc qu'il l'a touché.

Nous avons pas réussi à corriger cette erreur. Tout de même, le scorpion change d'état correctement et ses senseurs semblent fonctionner correctement. Après un pas de temps dt de 3s, le scorpion en mode MOVING change en IDLE et puis après un autre pas de temps de 5s, il change en WANDERING.

Puisque Madame Sam, nous a dit de nous concentrer plutôt sur la partie 5 que sur le debug de la partie 4.2, nous avons codé la partie 5. Nous avions juste eu des difficultés sur la compréhension des énoncés et de ce qui était exactement voulu de notre part. Une fois que nous avons saisi la manière d'implémenter les instructions et réfléchit à une conception, le codage était plutôt rapide. La seule erreur qui est apparue est que les deux statistiques, General et Waves, apparaissent en même temps. Nous avons seulement dû changer la méthode draw qui dessine que le graph qui a l'identifiant actif.

Nous avons aussi fait quelques extensions:

- Mushroom (quand on le mange on perd de l'energie)
- RIP (quand un objet meurt cela fait apparaître une tombe a cote)
- Banana (quand on va dessus cela nous desoriente)
- Fireball (un nouveau né est initialisé en boule de feu avec une vitesse plus grande que la norme pendant deux secondes)