Rapport

Beguin Mickaël - Hernandez Thomas

1 Présentation de la composante :

Dans le cadre d'un projet pour la 3ème année de licence informatique au CERI à Avignon, il nous a été demandé de réaliser la recopie du jeu Curve Fever, un jeu de type Snake en multi-joueurs. Pour ce faire, il a été décidé de partager les différentes parties du jeu en composante pour les élèves de la licence. Il y a ainsi 4 composantes: physique, graphique, réseaux et interface.

Nous avons décidé de travailler sur la composante physique. De manière générale, le moteur physique règle les problèmes d'ordre physique mécanique (collision, gravité, forces cinétique, etc..). CURVE FEVER est un jeu en 2D, ainsi, nous allons devoir gérer principalement le Snake, ses coordonnées, sa vitesse, sa position, etc. Les collisions entre les différents Snake ainsi que sur les bords du plateau. Ensuite, nous allons devoir gérer le plateau, le nombre de cases, l'occupation ou non d'un Snake ou d'un bonus. Et enfin les bonus, qui influencerons les états par défaut du Snake et du plateau.

2 Interactions avec les autres composantes :

A terme, nous voudrons que le jeu CURVE FEVER fonctionne, et soit jouable entre plusieurs joueurs humains sur un réseau local. Ainsi, l'interaction entre les différentes composantes devra être prise en compte dès à présent.

2.1 Composante graphique:

Le moteur graphique a besoin de connaître la position et la trace du Snake pour pouvoir l'afficher. Il en est de même pour les objets bonus/malus.

2.2 Composante réseau:

Le moteur réseau a, quant à lui, besoin des données physiques d'un joueur pour l'envoyer aux autres joueurs, et inversement, en réceptionnant les données physiques des autres joueurs.

2.3 Composante interface:

La composante interface fait appel au moteur physique lors qu'un round a débuté. Le calcul du score sera aussi une interaction entre l'interface physique et physique. Elle sera calculé en fonction du la mort ou non du Snake et si oui à quel moment. D'un autre côté, le moteur physique démarre quand l'interface utilisateur le demande (au début du round).

3 Formalisation

Pour mieux concevoir et formaliser le problème du moteur physique et ses interactions avec les autres composantes. Nous avons conçu un diagramme de classe UML pour identifier les différents éléments de la structure.

- La composante graphique aura un accès à la classe **Plateau** pour lire les données relatives aux coordonnées des **Snake**, et des objets (**FieldObject**, **SnakeObject**) afin de dessiner ces éléments.
- La composante interface aura besoin de la classe **Round** pour lancer une partie, et scruter la liste des **Snake** encore en vie et décider de la fin de la partie (ainsi qu'à l'id de chaque **Snake** correspondant à chaque joueur géré par l'interface).
- La composante Réseau quant-a-elle, doit synchroniser tous les éléments physiques chez tous les joueurs de manière continuelle, elle a donc un accès en lecture et en écriture à tous les éléments du jeu. Elle aura besoin d'un côté des informations d'un joueur (**Snake** du joueur, objets du **Plateau**), et d'un autre coté elle devra transmettre à ce joueur toutes les informations concernant les autres joueurs via le serveur. Elle mettra donc à jour les **Snake** et les objets (**FieldObject**, **SnakeObject**).

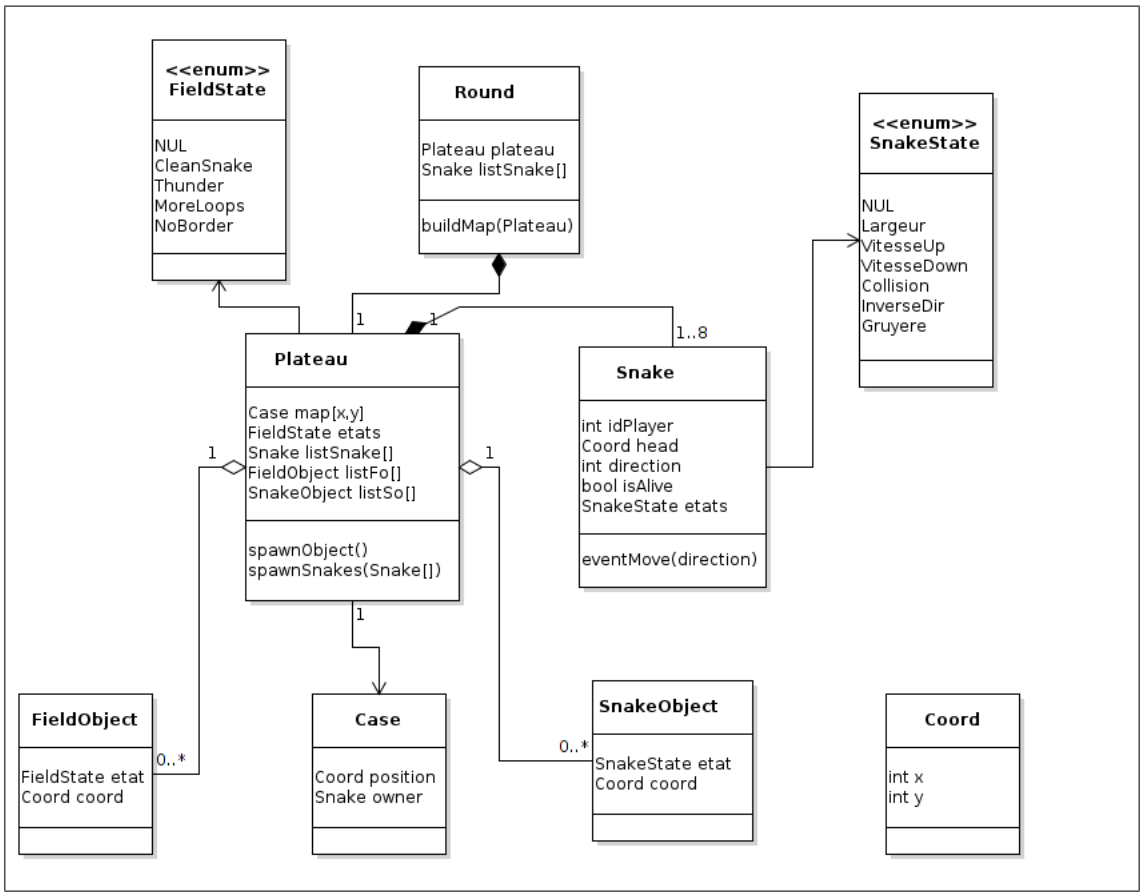

Figure 1: Diagramme de classes UML

3.1 Méthodes à invoquer depuis les autres composantes :

Pas de méthodes à invoquer depuis le moteur physique, il est autonome.

Remarque : La mise à jour des données des éléments physiques par le module réseau se fera de manière passive (c'est le module réseau qui prendra l'initiative de la mise-a-jour des données), il n'y a donc pas de méthodes à invoquer selon moi.

3.2 Méthodes à implémenter et invoqué par une autre composante:

La méthode buildMap() sera invoqué depuis l'interface utilisateur lors du démarrage du round, cette fonction aura pour but d'initialiser le plateau du jeu.