CoRe
Implémentation du patron chaîne de responsabilité

Antoine Friant
Christopher Meier
Daniel Palumbo
Lawrence Stalder
Valentin Finini

13 juin 2017

Table des matières

1 Introduction

CoRe est un projet dont l'objectif est de démontrer une implémentation du patron de conception "chaîne de responsabilité" dans une application ludique.

2 Le jeu

CoRe (pour "Chain of Responsability") est un jeu de type *shoot'em up* à défilement horizontal. Le joueur pilote un vaisseau qui avance à vitesse constante vers la droite, tandis que des obstacles à éviter viennent dans le sens opposé. Le joueur peut tirer des projectiles pour détruire ces obstacles.

Pour tirer un projectile, il faut appuyer sur les touches J, K ou L. Chacune de ces touches tire une couleur différente, et il est possible des mélanger ces couleurs pour en créer de nouvelles. Si on mélange les trois couleurs à la fois, la couleur résultant est le noir.

Le premier type d'obstacle est **l'astéroïde**. Afin de détruire un astéroïde, il faut tirer dessus avec la couleur qui lui correspond (un astéroïde rouge demandera un tir rouge). S'il explose, une réaction en chaîne (gérée par le patron chaîne de responsabilité) est déclenchée. Dans ce cas, l'objet le plus proche est lui aussi touché par la même couleur. Il se peut alors qu'ilexplose à son tour dans le cas ou la couleur est appropriée, et ainsi de suite ...

Le second type d'obstacle est **le bloc de glace**. Afin de détruire un bloc de glace, il faut qu'il soit touché par une réaction en chaîne de n'importe quelle couleur. Tirer dessus directement n'a pas d'effet. Lorsqu'il explose, il transforme la couleur reçue en blanc avant de continuer la réaction en chaîne. Le blanc détruit les astéroïdes de n'importe quelle couleur, ce qui a pour effet de faire exploser tous les astéroïdes de l'écran.

Les satellites, notre troisième type d'obstacle, ne réagissent que lorsqu'ils reçoivent du noir, lançant deux réactions en chaîne de couleur aléatoire (une chaude et une froide) en direction de l'objet le plus proche. Ils explosent après avoir reçu 20 tirs noirs.

Afin de ne pas rendre les réactions en chaîne trop rares, il a été décidé que les astéroïdes apparaissent par groupe de couleur chaudes ou froides. Il y a donc plus de chance d'avoir deux objets adjacents de la même couleur que si leur couleur était parfaitement aléatoire. La difficulté du jeu croît à mesure que le temps passe (de plus en plus d'obstacles apparaissent), et un score est affiché comme indicateur de la performance du joueur.

3 Implémentation

3.1 Libgdx

Libgdx [1] est un ensemble de librairies open-source et sous licence Apache 2.0. Il forme un framework minimaliste qui n'impose quasiment aucune structure ou classe au programmeur. Si le projet est correctement configuré, il est possible d'exécuter la même application sur desktop, IOS, Android et navigateur (HTML5).

Dans le cas de CoRe, il n'y a qu'une seule plateforme ciblée : desktop. La classe `DesktopLauncher` contient la méthode `main`, ainsi que les configurations de base pour lancer l'application (taille et titre de base de la fenêtre, par exemple). La boucle d'exécution est alors lancée.

La classe principale (dont le nom est laissé au choix du développeur, ici `CoRe`) contient la boucle d'exécution du jeu. La méthode `create` est appelée au démarrage, puis la méthode `render` est appelée à chaque rendu d'un image du jeu (60 fois par seconde par défaut). La méthode `dispose` est appelée à la fermeture du programme. Le développeur est libre de faire ce qu'il veut dans le corps de ces méthodes.

3.2 Structure générale

`EntityManager` est un singleton qui contient et gère toutes les entités affichables en jeu. Ses méthodes `updateAll` et `drawAll` sont appelées 60 fois par seconde, et consistent à mettre à jour et dessiner toutes les entités présentes en jeu en appelant les méthodes `update` (boucle logique) et `draw` (boucle graphique) de l'interface `Entity`. Toutes les entités implémentent donc `Entity`. Les entités pouvant faire partie d'une chaîne de responsabilité (réaction en chaîne) implémentent `ReactionHandler`.

`EntityManager` possède également un objet `Spawner` qui, à chaque mise à jour, génère des entités. A chaque fois qu'une entité est créée (par le `Spawner` ou même une autre entité), elle doit être ajoutée à la liste d'entités de `EntityManager`.

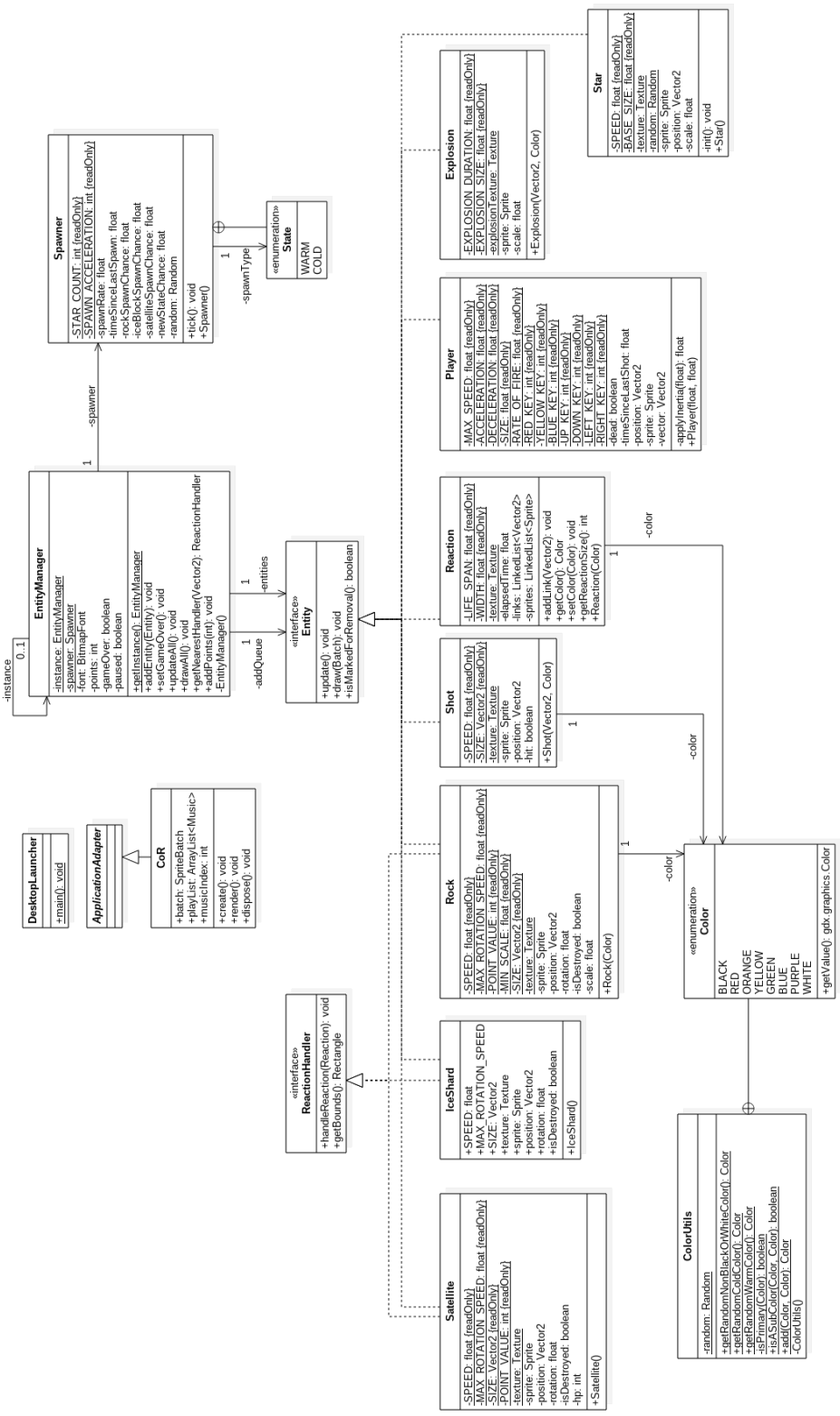
Certaines entités partagent des attributs et ont des comportements très similaires. Nous avons cependant décidé de ne pas factoriser le code en implémentant une superclasse pour deux raisons :

Premièrement, afin de proposer une bonne démonstration du modèle de conception chaîne de responsabilité, il a été jugé préférable de ne pas lier ces entités par héritage. De cette manière, nous disposons bien d'objets de natures différentes dans notre chaîne de responsabilité, plutôt que de n'avoir que des objets héritant d'une unique classe "`Obstacle`".

Deuxièmement, ces entités ne se ressemblent pas au niveau conceptuel. Un satellite tourne sur lui même de la même façon qu'un astéroïde, mais il pourrait aussi bien avancer en zigzag car rien ne lie leur comportement. Cette ressemblance est donc d'avantage le fruit du hasard qu'un lien de parenté entre ces entités.

1. https ://libgdx.badlogicgames.com/

4 Diagramme de classes



5 Patrons de conception

5.1 Chaîne de responsabilité

Les maillons de la chaîne de responsabilités sont les entités implémentant l'interface `ReactionHandler`. La requête qui est passée d'un maillon à l'autre est un objet de type `Reaction`.

`ReactionHandler` déclare les méthodes `handleReaction` (qui reçoit la requête) et `getBounds` (qui est nécessaire pour détecter une collision avec un élément de la chaîne). Lorsqu'un projectile entre en collision avec un `ReactionHandler`, il créé un objet `Reaction`, l'ajoute à l'`EntityManager`, puis l'envoie au `ReactionHandler` touché par le projectile.

La `Reaction` est passée de `ReactionHandler` en `ReactionHandler`. Elle peut être modifiée (longueur, couleur, aspect graphique), exploser/endommager un `ReactionHandler`, ou arrêtée (plus d'objet à l'écran ou couleur incompatible).

5.2 Singleton

Il a été décidé qu'`EntityManager` soit un singleton afin de permettre à n'importe quelle classe d'ajouter des entités à la boucle d'exécution du jeu. Elle permet également d'obtenir le `ReactionHandler` le plus proche d'une coordonnée spécifiée. La fin de vie d'une entité est signalée par cette dernière en renvoyant *true* à l'appel de `isMarkedForRemoval`. C'est une façon simple de créer des entités, les faire vivre et mourir, sans avoir à gérer de multiples boucles d'exécutions interdépendantes (comme c'était le cas lors d'une première implémentation).

5.3 Classe utilitaire

`ColorUtils` est une classe utilitaire permettant de manipuler des objets du type énuméré `Color`. Comme ces manipulations sont sans états, et que toutes les valeurs reçues et renvoyées sont des constantes, nous avons décidé de rendre le constructeur privé et de ne pas forcer le paradigme objet sur cette classe.

Chaque élément de ce type prend une valeur de type `com.badlogic.gdx.graphics.Color` (`Color` définit par libgdx). Cette abstraction permet de définir des méthodes sur ces couleurs indépendamment de la couleur réelle affichée à l'écran. Par exemple **BLUE** et **RED** font **PURPLE**, mais les couleurs affichées sont **CYAN**, **RED** et **VIOLET**. Grâce à cette abstraction, ce mélange de couleur reste vrai même si on change la valeur de **BLUE** de **CYAN** à **SKY**.

`ColorUtils` possède donc des méthodes statiques pour mélanger deux couleurs, renvoyer une couleur aléatoire, ou savoir si une couleur est contenue dans une autre par mélange (cette dernière méthode n'est pas utilisée dans la version finale du jeu).

6 Conclusion

Bien que l'implémentation d'une chaîne de responsabilité dans le jeu était l'objectif principal du projet, un autre patron de conception (singleton) s'est naturellement glissés dans le code, contribuant à la clarté de la structure du programme. Cette dernière est restée simple car le jeu lui-même est minimaliste et quatre autres projets requéraient également du temps de travail.