Berney Alec, Ferrari Teo, Forestier Quentin, Herzig Melvyn, Janssens Emmanuel

POO2 – 17.06.2021

Quokka’dventure

Modèle



Table des matières

[Mise en œuvre du modèle 2](#_Toc74845717)

[Modèle Commande 2](#_Toc74845718)

[Dans notre projet 3](#_Toc74845719)

[Retour en arrière 3](#_Toc74845720)

[Liste des actions 3](#_Toc74845721)

[Replay du niveau 4](#_Toc74845722)

[Diagramme de séquence pour l’exécution d’une commande 4](#_Toc74845723)

# Mise en œuvre du modèle

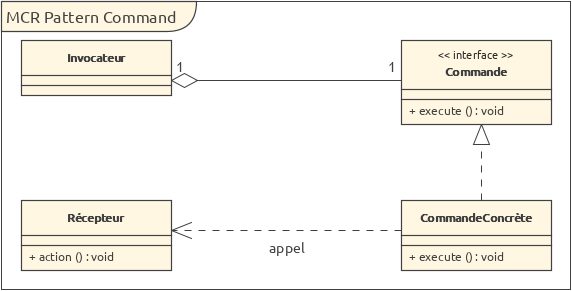
## Modèle Commande

Le modèle commande est un modèle comportemental permettant de séparer le code initiateur et l’action. Ce modèle permet une délégation de la responsabilité : une classe va déléguer une requête à un objet *commande* à la place de l’implémenter directement.

Ses objectifs sont de pouvoir envoyer des requêtes sans connaitre leurs caractéristiques ainsi que d’encapsuler lesdites requêtes. Il permet aussi d’éviter de dupliquer du code, de rendre les commandes sécurisées ainsi que de rendre le code plus facile à maintenir.

Les utilisations les plus connues de ce modèle sont : Gérer des actions venant d’une interface graphique ou d’entrées utilisateur, sauvegarder un état pour pouvoir y revenir, stocker les commandes pour pouvoir les réutiliser plus tard et créer des logs pour pouvoir réagir en cas de crash.

UML exemple du pattern :



## Dans Quokk’adventure

Dans notre projet, nous avons 3 manières visibles d’utiliser le modèle Commande : le retour en arrière (1.), la liste des action (2.) et le replay du niveau (3.).

Ces 3 utilisation reposent sur le fait que les commandes générées par l’utilisateur seront stockées dans un stack qui nous sert d’historique de commandes et vont pouvoir être réutilisées plus tard.



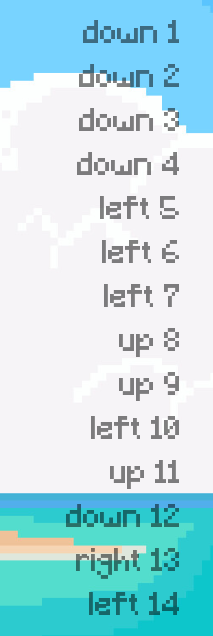
De plus, toutes les actions de l’utilisateur sont gérées par des commandes. Cela nous a permis, sans problèmes, d’avoir plusieurs entrées utilisateur comme les flèches cliquables sur l’interfaces graphique et les flèches du clavier.

### Retour en arrière

Le bouton de retour en arrière va permettre à l’utilisateur d’annuler sa dernière action.

Pour ce faire nous allons simplement aller chercher la dernière commande entrée par l’utilisateur, la dépiler de l’historique et utiliser sa méthode undo.

### Liste des actions

La liste des actions de l’utilisateur est affichée sur la droite de l’écran. Cette liste permet tout d’abord de visualiser clairement les commandes générées par l’utilisateur à chaque déplacement. De plus cette liste va permettre à l’utilisateur, en cliquant sur une des actions, de revenir en arrière, avant d’avoir exécuté cette commande.

Pour implémenter cette utilisation du modèle, nous allons tout d’abord afficher chaque commande, sous forme de bouton dans un scroller sur la droite de l’écran, lorsque celle-ci est générée. Quand l’utilisateur clique sur un des boutons, nous le dépilons et annulons toutes les commandes jusqu’à traiter la commande représentée par ce bouton.

### Replay du niveau

Quand l’utilisateur fini un niveau, Il a l’occasion de pouvoir revoir sa performance. En effet, en cliquant sur le bouton « Review » le niveau va se relancer et se rejouer automatiquement sans que l’utilisateur ne puisses interagir avec le déroulement de l’action.

Pour ce faire, nous créons un nouveau niveau qui va récupérer l’historique du niveau actuel et le parcourir à l’envers en exécutant chaque commande relativement à un timer. Procéder de cette manière va exécuter les commandes comme si l’utilisateur les avait générées sur le moment mais en utilisant les commandes stockées dans l’historique.

### Diagramme de séquence pour l’exécution d’une commande

Le point d'entrée peut être atteint de trois façons

* En appuyant sur les touches de claviers [W,A,S,D].
* En appuyant sur les flèches du clavier
* En cliquant sur les flèches affichées à l'écran

#### GameScreen

Scène principale du jeu

#### Tableau

Tableau d'acteurs qui vont être influencés par les commandes.

#### MoveComand

Une commande qui va déplacer un acteur.

#### PushCommand

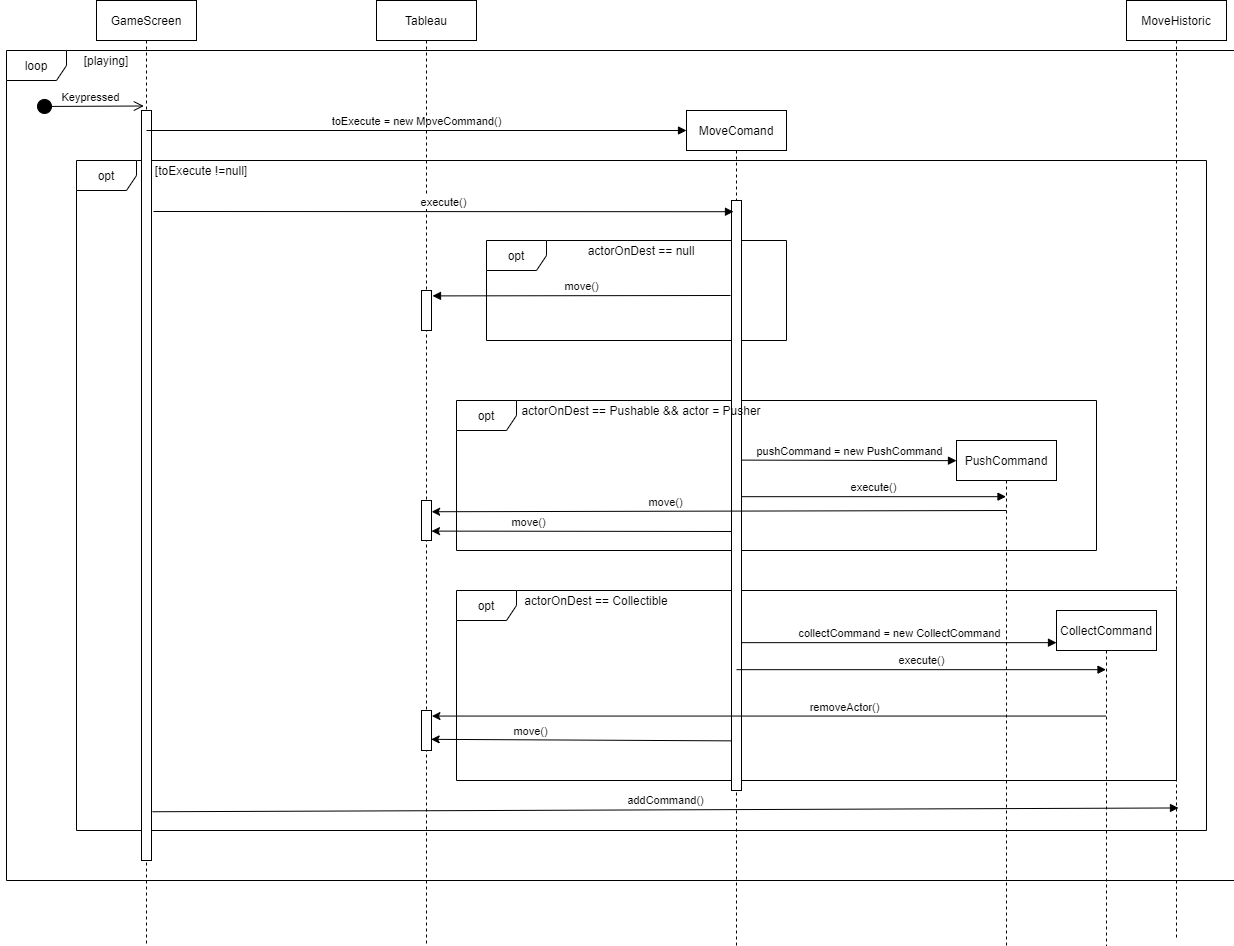
Une commande qui déplace un acteur uniquement s'il á été poussé par un autre acteur.

#### CollectCommand

Une commande qui va supprimer un acteur du tableau, et appliquer un effet spécial á l'acteur qui est passé dessus.

#### MoveHistoric

Permet de gérer/stocker un historique de commande pour pouvoir annuler une commande ou rejouer le niveau.



Lorsque nous annulons une commande, nous appelons la méthode *undo* de MoveHistoric qui se charge de dépiler la dernière commande et de l’annuler, s’il y en a une.