F.A.Q. Actionners/Mechanisms

◆ Comment fonctionne l'architecture globale?

Les Actionners et les Mechanisms sont répartis en deux grands ensembles :

1. Les WorldObject (pour les Actionners) et Les MonoBehaviour (Pour les Mechanisms). Ils sont présents dûs au fait que l'ancien système était directement dépendant à Unity.
2. Toutes les classes dans le namespace "interaction" (dossier interaction), qui sont indépendants d'Unity. Toutes les classes en faisant partie sont préfixés par "Interaction".

Les classes dans le namespace "interaction" sont elles-mêmes composées en trois sous-parties :

1. Les Mechanisms, qui reçoivent des messages et agissent en conséquence. Il peut s'agir d'une porte qui s'ouvre par exemple.
2. Les Actionners, qui agissent qui reçoivent des informations de l'extérieur et les envoie à des Mechanisms. Par exemple, un levier. Ils peuvent également recevoir un message de retour si un Mechanism se termine. Un Actionner peut recevoir plusieurs messages de retour sur un seul message envoyé, mais seul le premier actionner de la chaine recevra ces messages.
3. Les Modifiers, qui changent les informations du message alors qu'il est transmis d'actionner à Mechanism, ou de Mechanism à Mechanism (dans le cas où un Mechanism peut envoyer des messages).

Le fonctionnement global suit le principe du téléphone arabe, où chaque Mechanism ou Actionner correspond à une personne :

L'actionner envoie un message aux Mechanisms dont il a connaissance, après l'avoir éventuellement déformé. Le Mechanism écoute le message (après l'avoir modifié), interprète en actions, et le transmet (éventuellement modifié) à d'autres Mechanisms. Les modifiers correspondent aux déformations successives que subit le message au fur et à mesure qu'il est transmis.

Pour chaque Mechanism et Actionner, il existe 2 types de modifiers, ceux en entrée (qui correspondraient à l'ouïe pour le téléphone arabe), et ceux en sortie (La parole). Tous les Mechanisms appliquent les modifiers en entrées, les uns à la suite des autres de la liste, avant d'agir. De même, ils appliquent les modifiers de sortie un à un avant de transmettre le message aux autres Mechanisms.

Des mechanismProcess peuvent être créés pour exécuter un comportement sur la durée au sein des Mechanisms. Ils fonctionnent exactement comme des Process normaux, à l'exception faite qu'ils possèdent en plus un InteractionMechanism sur lequel agir, ainsi qu'un message (qui n'est PAS mis à jour automatiquement) qui lui sert de directives à suivre.

◆ Comment créer un nouveau Mechanism?

Pour ajouter un Mechanism, il faut :

1. Créer un script héritant de Mechanism qui s'occupera de faire les actions nécessaires.
2. Créer un script héritant de interaction.InteractionMechanism qui s'occupera de traiter les messages qu'il reçoit en overridant "`processMessage(CommandMessage cmdMessage )`". Il faut aussi penser à le rajouter dans le Mechanism qui en possède la charge et d'initialiser ses modifiers (`initModifiers()`).
3. Si il faut que le mechanism retourne un message vers l'actionner. Il faut utiliser la méthode `SendFeedbackMessage(CommandMessage cmdMessage).`

◆ Comment créer un nouvel Actionner

Pour ajouter un Actionner, il faut :

1. Créer un script héritant d'actionner.
2. Appeler les méthodes `interactionActionner.begin(CommandData cmdData)`, `update(CommandData cmdData)` et `end(CommandData cmdData)` au moment où l'on souhaite activer les mechanisms.

◆ Signification générale des CommandData des messages

`MainActivation`	Correspond à l'activation principale des mechanisms. Il est usuellement utilisé pour faire des actions analogiques.
`SecondaryActivation`	Correspond à l'activation secondaire des mechanisms. Ceux-ci ne change pas l'effet principal, mais rajoute un effet secondaire. Il est aussi utilisé pour lancer un cycle unique.
`FloatL`	Utilisé surtout avec MainActivation. Peut correspondre à la force appliquée par le bras gauche.
`FloatR`	Utilisé surtout avec MainActivation. Peut correspondre à la force appliquée par le bras droit.
`Power`	Rend le mechanism activable ou désactivable (comprendre, interactible). Cela ne change pas les comportements exécutés avant. **Un Mechanism qui reçoit un message avec Power à faux s'éteindra avant de traiter le message.**

◆ Comment trouver rapidement la source des problèmes d'actionner/méchanisms en intégration

Voici quelques étapes permettant de vérifier plus rapidement le fonctionnement des Mechanisms :

1. Notez le Mechanism ou l'actionner qui ne fonctionne pas correctement (Mechanism qui ne s'enclenche pas, etc...) . Souvenez-vous ensuite des paramètres du commandMessage qui auraient dû le faire marcher correctement (Par exemple, MainActivation et Power).
2. Regardez le message envoyé par l'actionner (le message de l'actionner lui-même, et les modifiers qui le change).
3. Ensuite, traverser chaque chaînon allant vers l'élément défectueux, en appliquant à chaque fois les modifiers en entrée et en sortie sur le message. Si l'arborescence est complexe, il pourrra être utile de noter quel message est transmis.
4. Il faut ensuite comparer le message que reçoit l'élément défectueux. De là, 3 cas possibles :
   a) Si le message transmis est incorrect par rapport aux paramètres nécessaires au point n°1, il faut manipuler les Actionners et les Mechanisms ainsi que les modifiers qui altère le message pour que ce dernier soit bon.
   b) Sinon, si l'élément défectueux est l'actionner, il faut regarder quand et quel message de feedback est transmis pas le Mechanism, ainsi que la façon dont l'actionner l'interprète.
   c) Sinon, cela veut dire que le Mechanism n'agit pas correctement selon le message résolu.