Presentation

This folder contains a mechanism system that I have created during my internship at Folimage on the game “The 4th planet : Drone Operator”, between May and July 2016.
The given code has not been modified since the end of the internship to better reflect the work made. Also, these are only code snippets, so they may miss some classes to function correctly, like some important parts of the core architecture which also belong to Kevin Silliam and Kevin Menuge, the other two developers of the game.

It is a system that allows an ActionnerUser (controlled by the player) to interact with game Actionners -like levers, buttons, scanners...-, and to activate different Mechanisms : Doors, Elevators and so on.
Also, mechanisms and actionners can be chained to create more complex behaviours.

Since the game has different ActionnerUsers with their own way of interacting with the environment, several actionners which can be interacted with and different interactible mechanisms, it needed to be as polyvalent as possible.

More information (in French) can be found in the documentation folder. It is the documentation made during the internship to help with the creation of new content.

Structure

The core system is independent of the game engine that was used (Unity) and are in the namespace interaction. More specific code are bound to derived Monobehaviours and are then bound to Unity.

There are 3 main parts in the architecture : InteractionActionners, MessageModifiers and InteractionMechanisms. InteractionActionners are used within Actionner monobehaviours, like InteractionMechanisms with Mechanisms.

1. Actionners

Actionners are the direct link between the player character and mechanisms. It interprets the player's actions, convert them into CommandMessages and send them to their known mechanism.

There is only one type of InteractionActionner, and specific actionners use it their own way.

2. MessageModifiers

CommandMessages are modified by MessageModifiers, which changes each of the message value according to its behaviour. For instance, it can set a specific value, change the message according to conditions, and so on.

3. Mechanisms

Mechanisms react to CommandMessages, so that it allows to do certain things based on the content of the message.

All mechanisms can be deactivated and rendered not interactible with the "power" parameter of the commandMessage. An inactive mechanism does not treat any message parameter until it has been reactivated again.

To allow direct interactions from the mechanism current state to the player character, a message is sent back to the actionner. For example, if a door has finished to open up, the lever activating it can be blocked, and the player freed from it.