Ecole Polytechnique Universitaire
de Lille

Projet IMA3

Rapport Projet SimulPhys

Corto Callerisa
Alex Lagneau
Clément Hue

Sébastien Dardenne
Quentin Normand

Soutenance le 4 juin 2019

Table des matières

Introduction

Dans un contexte économique où pour chaque entreprise, réduire les coûts de conception et de production est nécessaire afin d'être compétitif sur le marché, la simulation physique sur ordinateur s'impose comme un outil extraordinaire pour faire des premiers prototypes sans avoir à dépenser une fortune en crash-test. C'est dans ce contexte que nous avons choisi de réaliser un simulateur configurable de processus physique destiné à la conception et au développement de systèmes automatisés, s'implémentant aussi dans le concept d'entreprise 4.0.

1 Analyse du projet

Tout d'abord, nous avons dû définir les limites du sujet. C'est pourquoi dans un premier temps nous avons recherché quelles sont les solutions existantes (RobotDk, Delmia...) et étudié les différents systèmes présents à Polytech (Ascenseur, Tri de colis, Train 2D...).Nous avons ensuite choisi un système à reproduire (l'ascenseur) afin de le modéliser dans notre application.
Les objectifs sont alors de modéliser l'ascenseur avec une visualisation en 3D, et de permettre à ce modèle de communiquer avec un terminal distant.

Nous avons étudié plusieurs possibilités pour effectuer la modélisation : Unreal Engine, Unity, Godot,... Chacun présentant avantages et inconvénients. La facilité de prise en main et la portabilité sur différentes plateformes nous ont fait choisir Godot. De plus, Godot ne permet pas seulement une visualisation, mais possède également un moteur physique intégré performant nous permettant désormais de se concentrer sur la réalisation du modèle et sur la communication.

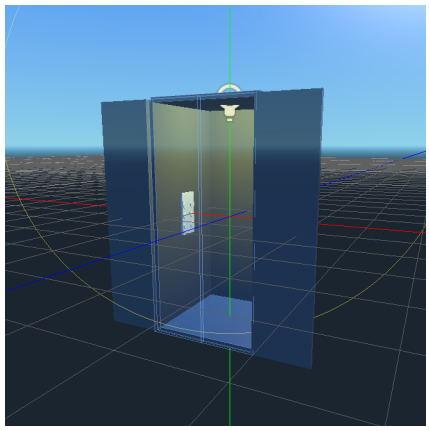
Nous avons également étudié différentes méthodes de communication : mémoire partagée, Pipe, TCP/IP, UDP... Nous avons choisi le protocole UDP car il est simple d'implémentation ce qui permet d'avoir une commande sous plusieurs formes (Godot, python, netcat sous linux). De plus, il est rapide ce qui permet un contrôle précis de la simulation..

2 Simulation

Nous avons décidé de simuler pour notre premier système un ascenseur. Le système possède 6 niveaux, est contrôlé par des boutons situés dans l'ascenseur et sur chaque palier pour l'appeler et comporte des capteurs pour le situer. Il est constitué des objets suivants : une cage d'ascenseur avec des portes coulissantes, des capteurs de présence et des boutons poussoir.

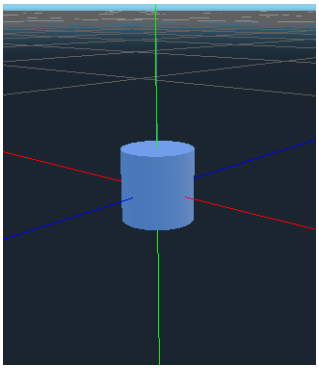
2.1 La cage d'ascenseur :

Le contrôle de l'ascenseur se fait par état : l'appui sur un bouton le met sur l'état "haut" ou "bas" pour le faire monter ou descendre ; les capteurs de présence lui indiquent à quel étage il se situe puis il change son état sur "stop" quand il arrive à l'étage désiré. L'ascenseur est géré par le script "CageAscenseur.gd" qui contient la fonction _process servant à modifier sa position dans l'espace en fonction de son état.



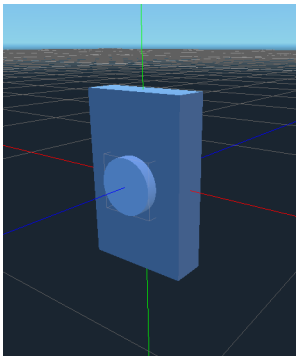
2.2 Les capteurs :

Les capteurs utilisent l'objet RayCast de Godot, cet objet peut détecter si un objet est positionné devant lui grâce à sa fonction "is_colliding()". On peut donc savoir quand l'ascenseur passe devant et, grâce à la fonction "update" de l'ascenseur, il peut lui indiquer à quel niveau il se situe. Si le capteur actionné est celui de l'étage désiré, la fonction update change l'état de l'ascenseur sur "stop".



2.3 Les boutons :

Le système est composé de 6 niveaux donc l'ascenseur contient 6 boutons et à chaque niveau se trouve un boîtier contenant un bouton pour appeler l'ascenseur. A l'appui du bouton le script appelle la fonction "appel" de l'ascenseur avec comme paramètre l'étage de destination ; l'ascenseur va modifier son état en fonction de l'étage appelé.



En plus d'effectuer ces actions, chaque appui sur un bouton ou changement d'état d'un capteur est notifié dans la commande et peut être affiché dans une invite de commande.

3 Communication

Après avoir étudié différentes méthodes de communication nous avons choisi d'utiliser le protocole UDP. En effet ce protocole permet une communication en réseau rapide ce qui permet une commande très réactive. De plus cela permet d'avoir une interface uniforme pour se connecter à la simulation, les bibliothèques UDP étant intégrées dans de nombreux langages.

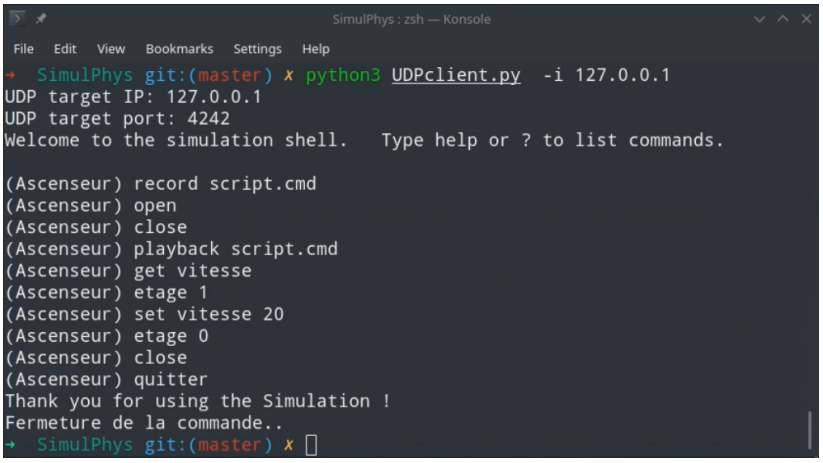
La communication se fait entre 3 parties indépendantes s'inspirant du modèle MVC (Modèle-vue-contrôleur). La commande s'effectue depuis un script avec une invite de commande, la vue est la fenêtre de simulation de Godot qui héberge aussi le modèle.

3.1 Transmission :

Nous avons défini et implémenté un protocol de communication simple :
'+-*/' <var> value : permet respectivement d'augmenter, diminuer, multiplier ou diviser une variable par une valeur.
get <**var**> **:** demande la valeur courante de la variable <var>
set <**var**> **value :** modifie de façon absolue <var>

Il y a aussi des fonctions spécifiques à la simulation de l'ascenseur :
etage <**num**> **:** permet de déplacer l'ascenseur à l'étage <num> compris entre 0 et 5.
open : ouvre les portes de l'ascenseur
close : ferme les portes de l'ascenseur

Il y a également des fonctions utilisées par la commande :
record<**nom_script.cmd**> **:** démarre l'enregistrement des commandes dans le fichier <nom_script.cmd>
playback <**nom_script.cmd**> **:** rejoue la séquence de commande sauvegardée dans <nom_script.cmd>
Le port et l'adresse ip du serveur de destination sont des arguments optionnels du script.
Il est possible d'ouvrir plusieurs fenêtres de commandes sur des postes différents.
L'implémentation d'un terminal spécifique à la commande augmente l'ergonomie pour l'utilisateur grâce aux fonctions d'autocomplétion et d'historique.

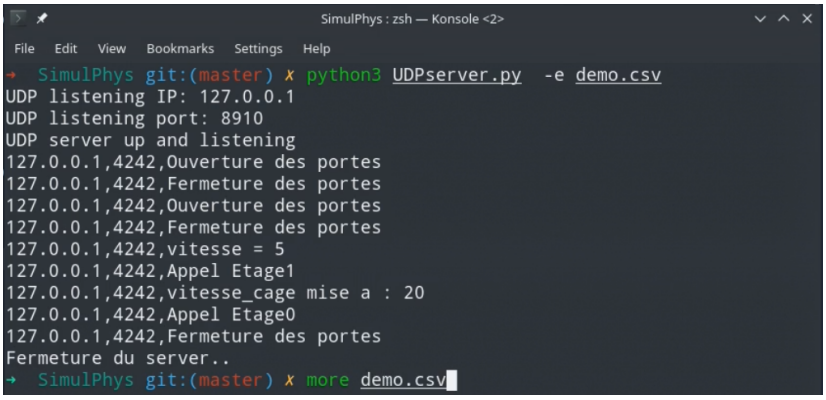


3.2 Réception :

Les messages envoyés par la commande sont reçus et traités dans la simulation par Godot.
A chaque frame le programme scrute l'arrivée d'un message et si un message est reçu celui-ci est "tokenizé", c'est à dire décomposé en "tokens" permettant de comprendre la commande. Exemple : la commande "+ vitesse 10" contient les tokens '+' 'vitesse' et '10'. Le programme peut donc comprendre qu'il doit augmenter la vitesse de l'ascenseur de 10 unité.
La simulation est ensuite mise à jours et un message est envoyé à la partie réception. Il est également possible d'interagir avec les boutons de la simulation pour appeler l'ascenseur. Un message indiquant la provenance et l'étage d'appel est envoyé.
Un menu permettant de configurer les paramètres d'envoi et de réception est inclus pour l'ergonomie utilisateur.

3.3 Réponse :

Le système est surveillé en temps réel et renvoie son état par communication UDP à chaque réception de commande. Ces réponses sont affichés dans une console et peuvent être exportés en format CSV si l'argument -e ou –export est passé au script.
Le temps est ajouté à chaque ligne avec une précision d'un millionième de seconde afin de faciliter l'analyse de la réponse du système.
Il est possible d'ouvrir une fenêtre de réponse sur plusieurs postes simultanément.



Conclusion :

Au cours de ce projet nous avons développé nos connaissances en informatique, particulièrement en modélisation et réseau, en réalisant un simulateur de processus physique en collaboration.

Nous avons procédé en suivant une démarche de projet. D'abord nous avons défini le sujet en précisant le besoin, ses limites et l'environnement d'implémentation. Nous avons ensuite fait une recherche documentaire afin d'étudier les solutions possibles pour la communication et la simulation ainsi que l'interface. Nous divisé le travail en deux grandes parties : Simulation et Commande. Finalement nous avons implémenté un simulateur d'ascenseur en Godot, avec une commande séparée codé en python et utilisant le protocole UDP.

Pour la suite notre objectif est d'implémenter un système plus complexe.