Douchin Julien
Jard Bastien

Rapport Projet Drinking Factory

Dans ce projet, nous avons développé tous les points du MVP ainsi que les options de l'avancement de la préparation, la détection des gobelets et l'ajout de la soupe dans la machine à café.

Les deux FSM :

Nous avons implémenté deux FSM dans notre projet : la première, qui s'appelle *drinkingMachine* permet de gérer le fonctionnement global de la machine à café et la deuxième, qui s'appelle *recette*, permet de contrôler la préparation des boissons. Bien que l'ensemble des fonctionnalités des deux FSM peuvent être rassemblées dans une seule FSM, nous avons choisi de les séparer afin de faciliter la lecture et le développement des fonctionnalités tout en ayant conscience que le fait de rajouter une seconde FSM augmente de façon considérable la taille de la solution finale de notre projet .

Gestion du choix de boisson et du paiement :

Afin de permettre une maintenance de code plus aisée, nous avons utilisé un seul état dans la FSM *drinkingMachine* pour simuler le choix d'une boisson. L'avantage principal est que nous n'avons donc pas besoin de modifier la FSM si nous voulons ajouter ou changer des boissons dans le distributeur. Cependant, un tel choix ne nous permet pas de savoir qu'elle est la boisson sélectionnée dans la FSM et doit donc être gérée du côté du code. Pour minimiser ce problème, nous avons ajouté un booléen dans la FSM qui nous permet de savoir si une boisson est sélectionnée ou non.

Le paiement se fait en 2 étapes distinctes : soit on choisit de payer en liquide soit par NFC sachant que nous avons choisi de ne pas permettre de mélanger les deux types de paiement. La condition pour que la préparation de la boisson se lance est qu'une boisson soit sélectionnée et que le paiement a été effectué. Dans notre cas, il faut donc passer le booléen à True si une des 2 conditions est réunie. Ce booléen nous oblige ici à sortir des états parallèles sans atteindre l'état final dans chaque état parallèle.

Un avantage de cette implémentation est que l'on peut ajouter d'autres états parallèles (comme par exemple celui qui permet un reset après 45s d'inactivité) qui n'auraient pas d'état final.

Gestion du déroulement d'une recette :

Pour pouvoir gérer le déroulement d'une recette et la préparation d'une boisson, nous avons choisi de développer la deuxième FSM *recette.* Pour cela, nous avons choisi de ne pas représenter toutes les étapes possibles à la réalisation des différentes recettes mais

de les regrouper sous des états globaux représentant à chaque fois une étape N de la recette.

Ce choix a pour but de privilégier la maintenance du code lors de l'ajout ou la modification d'une boisson. En effet, pour rajouter une boisson possédant un grand nombre d'étapes (par exemple une boisson avec 7 étapes), alors il suffit de rajouter un nombre d'états, avec le même nombre de variable pour représenter le temps, nécessaires afin d'atteindre le nombre d'étapes souhaités (actuellement 5 états sont utilisés et il suffira de rajouter uniquement 2 états et 2 variables pour ajouter la recette de la boisson du côté de la FSM).

Mais avec ce choix, nous ne pouvons pas interroger la FSM pour connaître le détail de l'avancement de la préparation de la recette. De la même façon, nous avons implémenté les options dans le code et non dans la FSM car nous ne pouvions savoir à quel moment les ajouter à la préparation sans connaître l'avancement de la recette.

Gestion du stock et de son rechargement :

La gestion du stock se fait uniquement dans le code : en effet, nous avons choisi de le gérer de cette façon pour pouvoir charger l'état du stock au lancement de la FSM dans le cas où le distributeur serait en panne ou hors tension. Nous avons donc un fichier data.xml qui permet de garder en mémoire notre stock pour chaque boisson ainsi que du sucre (de même que les informations clients pour les réductions au bout de 10 paiements). Cependant, un inconvénient majeur de cette solution est qu'il nous faut à chaque changement de boissons dans le distributeur, penser à aller gérer son stock dans le code pour que le fichier le prenne en compte à son tour.

En revanche, cela simplifie la lecture de notre FSM qui ne prend en compte que le bouton pour recharger les différents stocks lorsqu' une boisson n'est plus disponible (son bouton se grise dans ce cas là et le slider du sucre ne propose que les quantités disponibles si le stock ne permet pas d'ajouter autant de sucre que ce que le slider le permet en une boisson).

V&V :

Nous avons choisi de vérifier certaines propriétés dans notre implémentation des commandes, notamment pour le choix du paiement et de la boisson simultanées :

Sous le fichier 'testPaiementEtBoissonOk.lts' :

En utilisant la propriété de "fair choice for LTL check" de LTSA, nous avons vérifié que le client puisse choisir de payer ou de commander sa boisson en premier, et qu'à un moment donné il finira par avoir sa boisson correspondant à la catégorie Liveness Vérification pour notre système.

En utilisant la propriété de "unfair choice for LTL check", nous avons validé le fait qu'une fois le paiement effectué ou une fois boisson la boisson choisie, le client ne puisse pas changer de paiement en cours de route.

Sous le fichier ‘testErrorCase.lts’ :

En utilisant la propriété de “fair choice for LTL check” de LTSA, nous avons montré que si les choix sont plus nombreux et que l’on donne la possibilité au client de mélanger les 2 modes de paiements, alors on tombe dans un cas d’erreur. Ici, le check LTL nous retournera une “safety violation”, qui est le comportement attendu ici.

Prise de recul :

Ce que nous aurions fait différemment, c’est principalement l’implémentation des recettes parce que nous nous sommes aperçu bien plus tard que cela nous imposait plus de contraintes que ce nous avions imaginé : nous aurions voulu notamment implémenter les options dans la FSM et non autant dans le code que le montre notre solution actuelle. Pour cela on aurait dû intégrer les boissons différemment dans notre FSM pour avoir plus d’informations concernant le choix du client et ainsi modifier les sliders et les labels depuis la FSM directement.

De plus, le fait de profiter d’une réduction de paiement est actuellement dans notre code pour être réutilisé avec notre fichier de données mais nous aurions aimé avoir ce comportement dans la FSM directement.