Rapport Drinking Factory

Yann LEON Martin BOUTEILLER

Introduction

Dans ce projet nous avons d'abord implémenté au mieux la partie MVP à laquelle nous avons ensuite ajouté la détection des gobelets car le respect de l'environnement doit être une priorité. Puis afin de combler au mieux les envies de boisson des potentiels utilisateurs de notre Drinking Factory nous avons ajouté la gestion de la soupe et de l'iced tea. Puis une fois tout cela correctement développé nous avons terminé en implémentant la barre de progression des recettes pour avoir un retour sur l'avancement de la commande réalisée.

Choix de structuration

Au début du projet nous avons fait l'erreur de sous estimer l'utilité de la partie Yakindu et nous avons privilégié l'implémentation de notre Drinking Factory directement dans le code Java. Grâce à vos remarques nous avons refactor tout notre projet afin de réaliser nos développements de fonctionnalités en FSM directement, et prendre du recul afin de faire des compromis plus réfléchis dans notre implémentation.

La décision de lancement de la préparation

D'abord, la vérification qui valide une commande une fois un choix fait et le montant suffisant payé se fait par une synchronisation en sortie des régions "choice" et "payment" dans notre premier grand état orthogonal appelé "Main". Grâce à l'orthogonalité et la synchronisation on garde un grand dynamisme quant à la vérification du choix par rapport à la monnaie déjà payée par l'utilisateur, la mise à jour du choix et du paiement se font bien en parallèle. Lorsqu'il correspondent, on entre alors dans un état "Start" qui dans sa transition entrante aura sauvegardé l'état du choix et des options qui ont validés les conditions précédentes afin que l'utilisateur ne puisse plus modifier sa commande peu importe à quels boutons il touche pendant la préparation.

La condition présente en sortie de l'état "Paid" prend en compte les options potentiellement cochées par l'utilisateur pour vérifier qu'il paye bien les suppléments correspondants. Nous avons donc choisi de représenter les options à cocher non pas par un booléen qui pourrait être un réflexe mais par un entier qui prend la valeur 0 ou 1. Ce choix nous permet donc de vérifier le prix de la commande directement dans la FSM (ce qui est aussi valable pour le placement ou non de sa propre tasse). Ce n'est pas un choix très élégant que nous avons fait ici mais pour le peu d'options

à prendre en compte, nous avons préféré la simplicité et la réactivité de cette implémentation.

Les stocks

Les stocks des différentes recettes sont stockés dans la FSM sous forme de variable. Ainsi, lorsque une recette est lancée, c'est la FSM qui a la responsabilité de décrémenter le stock. La partie Java ne récupère cette variable seulement pour indiquer le restant des stocks à l'utilisateur ou pour l'empêcher de commander un produit indisponible (en grisant les boutons notamment). Cette solution responsabilise la FSM et donc présente comme avantage une sécurité. En effet, comme c'est la FSM qui détermine le stock des recettes, elle est sûre d'avoir toujours le bon chiffre et ne va jamais lancer une recette indisponible par exemple.

Les recettes

Les recettes ont été la partie la plus remaniée lors de ce projet. En effet, nous avions d'abord prévu de gérer les recettes dans le code métier, nous nous sommes rendu compte à temps que cela n'aurait pas été une bonne idée en raison du nom de la matière enseignée.

Ensuite, les états des recettes étant relativement similaires pour les boissons du MVP, nous avions fait des états uniques pour chaque étape de préparation que nous avions reliées ou non à l'arbre de la recette correspondant avec des gardes booléennes sur nos transitions. Une fois les recettes de la soupe et de l'iced tea ajoutées, nous nous sommes rendu compte que cette implémentation ne serait pas très viable pour un large choix de boissons. Son avantage premier était d'avoir un temps de préparation fixe pour chaque étape (que la vitesse d'écoulement de l'eau soit la même pour toutes les boissons et ne pas avoir à changer cette valeur sur de multiples transitions en cas d'ajout d'un robinet plus ou moins performant).

Enfin, pour palier au manque de lisibilité tout en centralisant les temps de chaque étape nous avons mis à profit les variables de notre FSM, ainsi nous avons réalisé un arbre d'étapes de recette propre à chaque boisson (ce qui implique de multiplier les états, par exemple celui qui simule l'ajout du sucre sera présent 4 fois) mais en ajoutant une variable de type entier pour chaque type d'étape que l'on reporte ensuite dans chaque transition correspondante. Le temps de chaque étape est donc centralisé dans la partie variables de Yakindu et cette méthode allie la lisibilité d'un arbre d'exécution clair pour chaque recette et une flexibilité dans la gestion du temps de chaque étape puisqu'il suffit maintenant de modifier la variable définissant le temps de transition des états correspondants.

En raison de la complexité de notre gestion des recettes intermédiaire il nous a semblé important d'implémenter des fonctions internes à notre FSM afin de vérifier le type de recette commandée, ainsi nous avons fait des fonction isXXX() pour chaque boisson et donc naviguer plus facilement dans notre arbre de recette avec

de nombreuses transitions enchevêtrées. Une fois le refactor de la gestion des recettes implémenté ces fonctions ont perdu un peu de valeur mais nous avons décidé de les garder car cela ajoute quand même un peu de lisibilité.

La synchronisation des étapes de chaque recettes

Dans chaque recette certaines actions sont réalisées en parallèle ainsi nous avons encore utilisé les états orthogonaux et les synchronisations. A cause de l'implémentation des synchronisations de Yakindu il n'est pas possible de synchroniser des transition avec une condition dépendant du temps et ne se valident pas au même moment. Pour remédier à cela nous avons dû ajouter des états WaitXXX avant la synchronisation avec une transition sortante marquée "always" pour réussir à valider la synchronisation.

Actions lors de la préparation

Dans le sujet il est spécifié que "aucune action ne peut être effectuée pendant la préparation d'une recette" et nous l'avons implémenté de la façon suivante : les boutons et sliders restent cliquables mais nous sauvegardons un état de la commande dès le lancement de la préparation donc chaque action faite par l'utilisateur n'aura aucune incidence sur la préparation en cours.

NFC et réduction

Le paiement par NFC est entièrement géré dans la partie Java le code entré par l'utilisateur est hashé et enregistré en tant que clé dans une HashMap ayant valeur un objet "Info" gardant en attribut le nombre de commandes effectuées et la somme des paiements antérieurs afin de calculer la réduction.

Pour la réduction de la 11ème commande, nous avons adopté une approche commerciale, c'est-à-dire que le client bénéficie d'une réduction de 1/10ème de la somme des 10 paiements antérieurs. Ainsi le client sera plus tenté de prendre une boisson plus chère que la moyenne de ses anciennes commandes afin de maximiser le profit apporté par la réduction plutôt que juste avoir une boisson gratuite à cause de laquelle il ne mettrait pas à profit la totalité de sa réduction.

V&V

Pour ce qui est de la Vérification et Validation nous n'avons pas réalisé de code LTS. Cependant, nous avons pensé à des questions intéressantes à poser à notre système.

Vérification

Les stocks sont-ils bien décrémentés à chaque préparation ? (liveness) : □(isTea()⇒teaStock--)

Validation

Lorsque le client donne son propre gobelet obtient-t-il une réduction de 10 centimes ? (liveness) : □(ownCup⇒discount)

La préparation ne doit pas se déclencher si le client n'a pas mis assez d'argent pour sa commande. (safety) : □(!startRecipe)U(money<price)

Prise de recul

Nos choix de départ nous ont handicapé sur le bon déroulement du projet. En effet à cause de ces derniers, nous avons dû à mi-chemin refactor l'entièreté du projet rendant alors le travail effectué lors des premières semaines inutile. Néanmoins le choix de plus responsabilisé la FSM est un choix meilleur, de part son efficacité, simplicité, sécurité et son respect des attentes de notre client.

Les points d'amélioration qu'ils nous semblent importants d'aborder dans cette prise de recul sont essentiellement liés à l'interface homme-machine. Effectivement compte tenu de notre temps limité suite à la rectification de nos erreurs, nous avons décidé de préférer la bonne avancée sur les fonctionnalités dans notre code quite à ne pas tout communiquer à l'utilisateur.

Ainsi l'utilisateur ne voit pas le prix de son choix avant d'avoir payé. De même l'affichage de la barre de progression est minimal, ne s'actualisant qu'à la fin des grandes étapes et non durant leur déroulement. Pour palier à ce problème nous aurions pu rajouter un every pour chaque étape de notre FSM. En effet, comme c'est la FSM qui connaît le temps de préparation de chaque étape, il était possible de rajouter : "every tempsDePreparation÷pourcentageDeLaBarreSouhaité s / raise add1percent".

Un problème que nous aurions pu rencontrer ici est le fait que every prends des int en paramètres donc implique un cast.Cependant un cast aurait alors entraîné des pertes lors de la conversion et donc des erreurs de quelques pourcent au final dans la barre de progression.

Nous sommes conscients de ces points d'amélioration. Malgré cela nous considérons que notre programme est viable et compréhensible pour le plus grand nombre.

Annexe

Un peu d’aide pour la prise en main de notre Drinking Factory :

- Pour récupérer sa boisson, il faut cliquer sur l’image du gobelet (ou la tasse).
- Pour le paiement NFC, on entre un code puis on clique sur le “biiip” (un utilisateur avec déjà 9 commandes et une somme des commandes passées de 4.50€ est déjà présent et est utilisable si on ne met pas de code).
- Pour les stocks, leur initialisation se fait en haut du constructeur de la DrinkingFactoryMachine (leur valeur s’affiche dans l’interface, soit à gauche pour les boissons, soit à droite entre crochets pour les options).
- Pour récupérer sa monnaie, il faut cliquer sur le montant de monnaie qui s’affiche en bas à droite.