Météreau Frédéric Marti Ludovic

Rapport de projet
Machine à café

Le projet final de notre machine à café contenant le code, le rapport et le dossier LTSA se trouve sur le RENDU.

Implémentation

- MVP

Nous avons implémenté l'intégralité des exigences 1 à 17 du sujet avec précaution afin de toujours répondre au mieux à la consigne. Nous avons toutefois pris quelques libertés si nous les estimions pertinentes et que notre expérience avec les machines à café nous le permettait ou bien lorsque le sujet ne précisait pas la chose.

- MVP n°3 : nous avons décidé de réinitialiser la commande courante en plus du paiement en cas d'abandon.
- MVP n°4 : les curseurs reviennent à un état initial après 45 secondes d'inactivité de l'utilisateur mais pas après une transaction. Nous avons décidé de garder leur position pendant la préparation de la recette et de ne les réinitialiser qu'une fois cette dernière finie. Cela permet d'avoir un accès simple à l'information transmise, pour la machine et l'utilisateur, à tout moment pendant la préparation de la boisson.
- MVP n°6 : nous avons décidé de n'afficher le prix de la commande que dans la console de l'UI afin de le rendre dynamique selon les options choisies ou autres modifications mais également dans un souci d'épuration de l'interface déjà très chargée avec certains textes ne pouvant pas toujours s'afficher intégralement sur les boutons.
- MVP n°7 : nous n'effaçons pas les informations d'identification de la carte servant pour la fidélisation NFC dans le champ prévu à cet effet afin de simplifier l'utilisation de l'interface. Aucune faille de sécurité bancaire ne devrait être à suspecter dans notre programme.
- MVP n°9 : nous avons décidé de nommer les tailles "S", "M" et "L" en lieu et place de "Short", "Normal" et "Long" pour alléger l'effort visuel et cognitif de l'utilisateur.
- MVP n°11 : nous rendons également la monnaie lorsque le paiement s'effectue finalement par carte bancaire.
- MVP n°13 : nous avons décidé de laisser la possibilité d'ajouter des pièces pendant la préparation de la boisson. Ces dernières seront bien sûr restituées à la fin de la préparation. Nous gardons simplement cette fonctionnalité comme "clin d'œil" car c'est une des premières implémentées et qu'elle n'impacte pas le reste de la machine.
- MVP n°14 : le mécanisme interne de la machine se nettoie automatiquement après chaque préparation finie ou avortée. En effet, si l'utilisateur décide de récupérer sa préparation avant que celle-ci ne soit finie (ce qui n'est pas conseillé de faire), la procédure sera interrompue sur le champ et le nettoyage débutera. De plus, si la préparation est finie, notre machine n'attendra pas que l'utilisateur ait récupéré son gobelet pour démarrer le nettoyage.
- MVP n°16 : nous avons considéré les ingrédients suivants : gobelets de taille unique, grains de café, dosettes de café, dosettes de soupe, sachets de thé, doses de sucre, doses de sirop d'érable, doses de lait, doses d'épices, doses de glace vanille, doses de croûtons et doses d'azote liquide. Ils sont initialisés en petites quantités afin de pouvoir constater du comportement rapidement (3 doses pour les ingrédients à quantité invariable).

- MVP n°17 : une commande rendue gratuite par fidélisation NFC ne sera rendue possible que si dix achats au moins ont été effectués avec la même carte depuis la dernière commande gratuite et si le prix de la commande actuelle ne dépasse pas la moyenne de prix des dix derniers achats les plus chers avec cette carte depuis la dernière commande gratuite.

- Extensions

Toutes les extensions proposées ont été implémentées dans notre machine.

- Soupe : l'implémentation est complète, la recette est suivie. Pour gérer les épices, comme pour les options qui demandent des modifications de curseurs, on a décidé de modifier directement le titre ou les valeurs de ce dernier.
- ice tea : même remarque que précédemment, on modifie les valeurs des curseurs "Size" et "Temperature" pour satisfaire l'exigence. Le prix selon la taille est bien géré dynamiquement et on a décidé d'ajouter une image de porte "FERMÉE" pour illustrer le verrouillage pendant l'application de l'azote liquide.
- progression : la progression de notre barre se fait pourcent par pourcent quelle que soit la durée de préparation. Cependant cette progression sera plus ou moins lente et correspondra toujours avec le temps d'attente ce qui nous permet d'avoir une progression fluide plutôt que "par blocs".
- gobelet : implémenté comme demandé.

Choix de conception

- Gestion du temps

Une des premières questions qu'on s'est posé est celle de la gestion du temps. En effet, nous voulions décider de cette implémentation au plus tôt comme elle nous servira tout au long du développement. Les choix que nous avons considéré sont les suivants : implémentation dans le code métier uniquement, implémentation dans la "statechart" (fsm) directement ou implémentation dans le code métier et dans la fsm. Nous avons de suite éliminé la première option pour des raisons simples. Tout d'abord, nous avons eu l'occasion de tester cette implémentation pendant les TD des semaines pré-projet et elle nous semblait trop "lourde" et "compliquée" pour une utilisation efficace. De plus, nous savions que l'utilisation du temps dans la FSM nous serait presque indispensable pour un rendu plus compréhensible et efficace (plus logique aussi car les transitions des différents états que nous avons ont presque tout le temps un rapport au temps). La troisième option aurait pu être un bon compromis pour certains cas nous permettant plus de souplesse dans l'utilisation de la FSM mais nous avons quand même penché vers la seconde car elle se rapprochait plus de la vision d'une FSM plus "complète" et "transparente" dans les processus de choix et fabrication de boissons que nous avions. De plus, avec cette implémentation on obtient une meilleure séparation des responsabilités de la FSM et du code métier.
D'où l'absence de gestion de temps dans le code métier de notre machine à café.

- Stockage des données

Nous n'avons pas beaucoup hésité pour cette question. Effectivement, notre première implémentation de FSM récupérait beaucoup de données après chaque action effectuée pour les stocker et les réutiliser sans appels au code métier. Le problème : on se retrouve vite noyé dans un très grand nombre de données dans la machine à force d'ajouts de fonctionnalités. On a alors commencé à stocker certaines informations dans le code métier pour alléger la FSM mais la centralisation des données permettait une gestion plus logique. Nous évitons ainsi un maximum de redondance et permettons l'ajout de boissons ou d'options plus simple sans ajouter de variables.
Pour cela, nous avons créé différents objets et types énumérés contenant toutes les informations d'une boisson dans le code métier, et la FSM en demanderait l'accès seulement si nécessaire. Enfin, cette implémentation nous permet de rester sur notre vision principale d'une FSM qui décrit les étapes de la machine à café de la façon la plus claire et compréhensible possible.

- Gestion des calculs

Quand on parle de calculs, il s'agit principalement de légers calculs pour connaître les différentes durées qui interviennent dans les étapes de fabrication du produit. On a opté pour l'utilisation d'opérations dans la FSM qui donnent directement le temps voulu. L'idée de récupérer les valeurs des curseurs notamment pour ensuite effectuer le calcul dans la FSM

n’est pas mauvais en soi mais comme ces valeurs sont connues du code métier et que des opérations dans la FSM sont ici quasiment inévitables (dans notre modèle ne comportant la gestion du temps que dans la FSM tout du moins), nous préférons donner un résultat déjà calculé, ce qui allège la charge de travail de cette dernière. Nous pouvons ainsi également ajuster et modifier les différents calculs en ajoutant des paramètres par exemple sans jamais avoir à modifier l’apparence de la conception de boisson dans la FSM (elle reste alors dans une forme de description des étapes et états successifs des recettes modulables dans leur durée mais pas leur forme).

- Gestion des stocks

Nous avons là un nouveau problème de stockage des données. Nous pouvions cette fois beaucoup plus simplement gérer ces informations dans la partie FSM du projet. Cependant nous aurions eu de nouveaux problèmes concernant l'interaction avec l’UI. En effet, la gestion des stocks demande de bloquer des boissons ou options selon la quantité en magasin. Donc en plus de charger la FSM avec une quantité de données proportionnelle au nombre d’ingrédients, les appels à des méthodes du code métier pour modifier l’interface selon les quantités restantes ou pour obtenir l’information des curseurs afin de connaître une quantité seraient aussi multipliés.
Finalement, nous pensons que les stocks devraient plutôt être un paramètre qui influence les possibles états de la FSM que des états propres.

- Fidélisation NFC

Comme pour les problèmes de stockage de données, la fidélisation NFC et l’enregistrement de nouvelles cartes de crédit complexifieraient beaucoup la FSM tout en nous éloignant de la vision compréhensible et fonctionnelle que nous en avons. En effet, ces ajouts demandent de créer un nombre d'objets potentiellement infini qui ne peuvent être stockés que depuis le code métier (stockage dynamique de notre implémentation). De plus, le reste de notre machine semblait tendre vers une gestion des prix liée à la gestion des données donc externe à la FSM.
Il nous a donc semblé plus naturel et évident, en accord avec ce qui était déjà implémenté, de s’occuper de la fidélisation NFC de façon totalement externe à la FSM.

- Exécution des recettes

Notre première version de la machine à café se voulait tellement simple et épurée qu’elle ne comptait qu’un seul état “recette”, dont toute l'exécution était faite dans le code métier. Cependant cette version ne permettait pas de voir explicitement les différents états lors de la fabrication dans la FSM et notre machine à café s’éloignait beaucoup trop de l’utilisation d’une machine à état. De plus, conformément à notre choix d’avoir un rapport au temps absent du code métier, il fallait voir les choses autrement. Nous avons alors décrit toutes les étapes de la recette et ses états intermédiaires sous forme de blocs dans la FSM.

- Organisation des blocs

Pour ne parler, maintenant, que de la FSM, la façon dont nous avons regroupé nos différents états n'est pas le fruit du hasard. Il est tout à fait possible d'envisager un fonctionnement sans états parallèles en augmentant légèrement le nombre de vérifications et en ayant des états plus complets. Cependant, bien que le fait d'effectuer deux tâches différentes simultanément serait toujours possible, il serait moins modulable et compréhensible d'un simple coup d'œil. L'idée derrière notre façon d'organiser les états et les blocs est de pouvoir retrouver la consigne dans l'architecture. Par exemple, pour l'exécution d'une recette, plusieurs actions se passent en même temps et sont donc regroupées dans le même bloc dans notre FSM (actions entre parenthèses séparées par des "|" dans l'énoncé). Les différents blocs s'enchaînent ensuite les uns les autres pour former la recette (séparation par des ";" dans l'énoncé).
Pour ce qui est des conditions de sortie des états orthogonaux, nous avons essayé toutes les idées que nous avions. Les deux plus pertinentes que nous avons retenu sont les sorties par conditions booléennes et par synchronisation. Ne sachant pas trancher entre ces deux options, nous les utilisons toutes les deux et elles répondent parfaitement à nos attentes.

V&V (LTSA)

- Présentation

Afin de réaliser des vérifications compréhensibles et simples, nous avons décidé de séparer notre fsm en plusieurs fichiers LTSA. En effet, notre fsm est décompensée en 5 fichiers LTSA. Lors du *check safety,* un DEADLOCK, ceci est expliqué par le découpage de la fsm. Lorsqu'une partie de la fsm arrive à la fin, le fichier LTSA se stop alors que dans la fsm, on passe à l'état suivant. Les questions posées aux systèmes vont vérifier les transitions pour être sûr qu'il n'y ait pas de réel DEADLOCK. La gestion du temps n'étant pas implémentée dans LTSA, nous avons traduit les transitions temporelles avec des transitions équivalentes non temporelles (Ex : *after 3s* pour les croûtons se transforme en *afterPouringCroutons*).

Fichiers LTSA :

- Order.lts
- HeatingAndCup.lts
- WaterAndSugar.lts
- Options.lts
- EndOfPreparation.lts

- Questions

Order.lts

Partie de la fsm correspondant aux choix des options, des sliders, de la boisson et du paiement.
Pour cette partie, nous n'avons pas réalisé de question car chaque partie de l'état parallèle global étant indépendante et dont certaines n'avaient pas d'impact sur l'évènement de fin, il était difficile de trouver un moyen de vérifier les actions avec des questions. De plus, nous n'avons pas trouvé le moyen de revenir sur un même état parallèle dans LTSA sans en recréer un et donc avoir une boucle infini.

HeatingAndCup.lts

Système qui vérifie le placement ou non du gobelet, du sachet de thé, de la capsule etc en fonction de la boisson sélectionnée ainsi que le chauffage de l'eau.

Pour chaque partie de l'état parallèle, nous avons une propriété liveness qui vérifie qu'au moins quelque chose se passe bien lors du passage dans cette partie comme par exemple le chauffage de l'eau, le placement de la dosette de café, le traitement des grains pour l'expresso. Nous avons aussi les mêmes propriétés mais en safety pour vérifier que rien de mauvais se passe.
Enfin, pour la partie du placement du gobelet, du sachet de thé ainsi que de la soupe, nous avons plusieurs autres propriétés safety et liveness, un peu plus compliqué en fonction des

choix faits. Par exemple, si nous choisissons un thé, il est nécessaire qu'une action correspondant à la soupe ne soit pas faite.

WaterAndSugar.lts

Fichier lst correspondant au versement de l'eau ainsi que du sucre, du sirop d'érable ou des croûtons.
Comme pour le fichier précédent, nous avons une propriété de liveness qui vérifie que l'écoulement de l'eau se passe dans tous les cas. Ensuite nous avons plusieurs propriétés de safety afin de vérifier les conditions entre le versement du sucre, du sirop d'érable ou des croûtons. Nous avons également des questions de safety pour vérifier que les actions sont bien effectuées en fonction du choix fait.

Options.lts

Partie de la fsm gérant les options de la vanille, du nuage de lait et du thé glacé ainsi que l'infusion du thé.
Pour cette partie, toutes les questions posées au système découlent de la vérification des conditions. En effet, à chaque choix possible lors d'une condition, des propriétés de safety et liveness sont faites pour vérifier le déroulement des conditions. Par exemple, dans le cas de l'infusion du thé, il n'est pas possible de pouvoir mettre de la glace, pour vérifier ceci, nous avons une propriété qui vérifie que lors de la sélection d'un thé, il n'y aura aucune action lié à l'option de la glace à la vanille.

EndOfPreparation.lts

Vérification du nettoyage de la machine et de la préparation récupérée.
Ce dernier fichier vérifie simplement à l'aide de deux propriétés de liveness que lorsque la préparation est finie, qu'on attendra que la machine soit nettoyée et que la préparation ait été récupérée.

Autoévaluation

- Répartition du travail

Sur ce projet, nous avons tous les deux travaillé de manière équitable en alternant les tâches. D'une semaine à l'autre, l'un de nous s'occupait du code métier pendant que l'autre travaillait sur la FSM. À la fin du projet, nous avons changé notre façon de travailler en se concentrant sur chaque fonctionnalité en particulier à tour de rôle.

- Critique

Beaucoup de choix ont découlé de décisions précédentes comme la gestion du temps ou celle des données qui impliquent de scrupuleusement séparer les responsabilités là où, parfois, une solution intermédiaire pourrait valoir la peine.
Pour ce qui est des données par exemple, une grosse partie pourrait très bien être contenue dans la FSM, ce qui, certes, la rendrait plus chargée et augmenterait beaucoup le nombre d'états et de transitions, mais qui la rendrait aussi moins abstraite. Cela permettrait en plus de poser davantage de "questions" à la FSM pour s'assurer de son bon fonctionnement par Vérification et Validation.

Un souci mineur constaté est celui de la structure de la FSM concernant certains états qui s'enchaînent parfois de façon chaotique sans vraiment appartenir à un sous-ensemble ou un bloc. Ce n'est cependant pas très gênant car on arrive toujours à suivre les différentes étapes d'une recette sans effort.

- Modifications éventuelles

Une modification qui nous semble très pertinente au premier abord est celle de notre modélisation "statechart". Comme nous l'avons dit plusieurs fois, notre idée première était de garder une FSM claire et compréhensible qui suit les étapes des recettes visuellement et fonctionnellement. Et même si cette vision nous a beaucoup guidé et a été respectée, elle aurait pu être mieux abordée. Si nous avions vu le sujet dans son ensemble dès le début au lieu d'avancer pas à pas, des meilleurs regroupements et distinctions entre certains états et leurs interactions auraient pu voir le jour.

Il en est de même pour le code métier. Nous n'avions pas conscience de l'importance de la structure de ce dernier et pour cela, nous avons négligé l'organisation des classes. Notre développement a tout de même progressé mais, comme dans tout projet, nous aurions gagné un temps considérable en adoptant une structure adaptée dès le début. Cependant, à force de retravailler l'architecture, nous nous en sommes très bien sortis avec une répartition des responsabilités suffisante mais elle aurait pu être plus précise et mieux répartie.

Finalement, une meilleure vue d'ensemble du projet dès les premières heures aurait été favorable et nous aurait permis d'avancer dans le bon sens plus tôt.