Rapport FSM - Drink Factory

Equipe

Timothée Poulain
Delmotte Vincent

2020-2021

POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

I) Fonctionnalités

a) MVP

Nous avons implémenté la version MVP de la machine. Nous pouvons effectuer un paiement par carte bleue ou en monnaie. Le paiement peut s'effectuer avant ou après la sélection de la boisson. En cas d'annulation ou après 45s sans action la boisson est annulée et on rembourse le client. Pour la carte bleue on retient les informations et on paie juste avant le commencement de la boisson. On peut annuler à tout moment le paiement avant la préparation de la boisson.

Le système de fidélité permet lors du 11ème paiement de réduire le prix de la sélection à hauteur de la moyenne de ce qui a été payé dans les 10 paiements précédents.

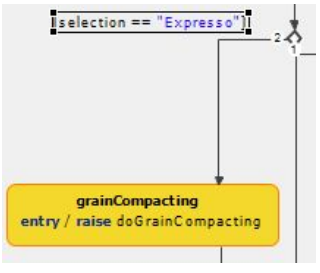
La sélection de la boisson et de leurs options se gère grâce à plusieurs boutons nommés. Il y a trois sliders qui permettent en fonction du type de boisson de sélectionner la taille, la température et les épices ou le sucre ajouté.

Une fois le paiement effectué, on entre dans la réalisation de la boisson. Toutes les étapes des recettes sont presque respectées, sauf pour l'expresso qui a une étape en moins (voir plus bas : problème Expresso)

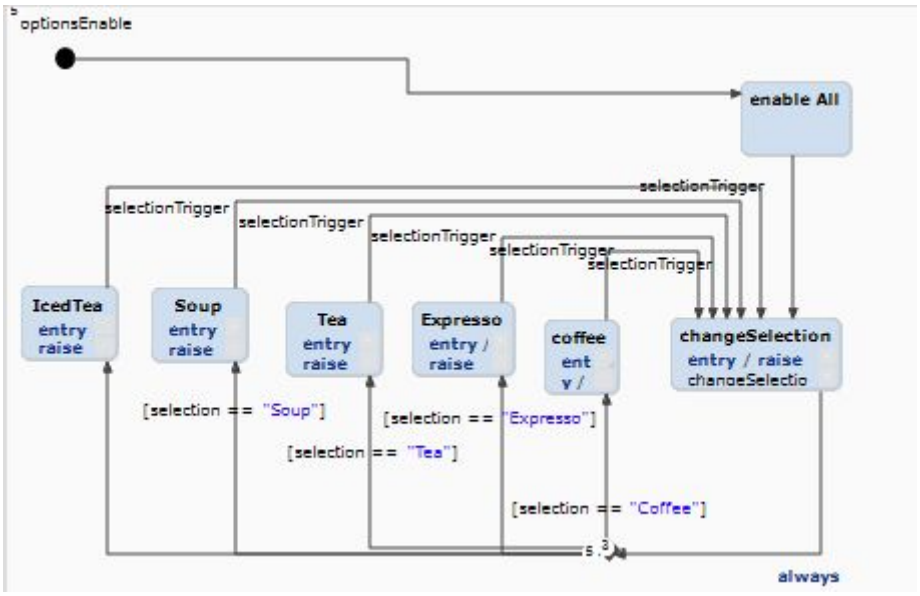
A chaque fin de préparation la machine est nettoyée et remise à 0 (reset). Les boissons en rupture de stock ne peuvent pas être sélectionnées par l'utilisateur. De plus, on ne peut plus influencer la préparation une fois que cette dernière est lancée.

Il est possible d'ajouter des options dans les différentes boissons mais ceci avant le paiement car dans le cas contraire le processus de réalisation se lance si le montant est suffisant.

Problème Expresso : Lorsque la FSM atteint l'état "Grain Compacting", la méthode *doGrainCompacting* n'est pas exécutée. Pourtant l'état est bel et bien traversé, comme le montre la capture d'écran ci-dessous. À ce jour, nous n'avons toujours pas identifié la cause du problème malgré une vérification du code de la méthode.



Problème de modification après le début de la préparation : Le MVP indique que lorsqu'une préparation est lancée, les options ne sont plus modifiables, ce qui est logique. Pour résoudre cette problématique, nous avons commencé à mettre en place un système de verrouillage des boutons, avec un parallel State pendant la sélection :

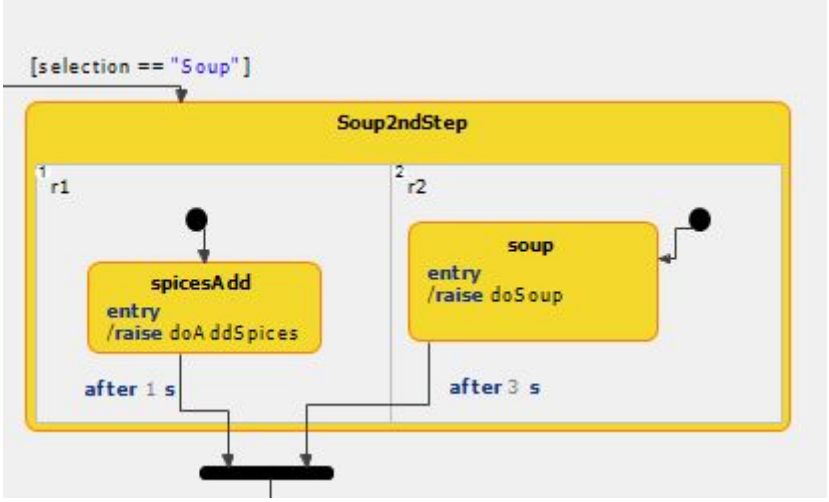


Cependant, des problèmes de compilation de Yakindu nous ont empêché d'implémenter les méthodes associées aux états, et nous n'avons pas eu le temps de le corriger (c'est la dernière modification que nous avons apporté au projet). Nous avons donc dû opter pour une solution avec un booléen qui n'est pas très propre mais c'est malheureusement l'option la plus adaptée dans ce cas.

b) Extension : Gestion de la soupe

La soupe apparaît dans la sélection des boissons. Le slider du sucre passe en mode épices. Le prix est respecté et la préparation adéquate est lancée une fois la face de sélection et paiement fini. Des épices sont ajoutées à la place du sucre si cette option est demandée.

Problème : à la fin de la chauffe, la FSM est supposée passer à l'étape suivante. Cependant, nous avons constaté que la FSM ne sortait jamais du bloc "Soup2ndStep", qui se charge d'ajouter les épices et la préparation de soupe dans l'eau. Après de nombreux tests (dont l'ajout d'un *fork* à l'entrée des *parallel state*), nous en ignorons toujours la cause. La préparation de la soupe n'arrive donc malheureusement jamais à terme.



c) Extension : Gestion de l'ice tea

L'ice tea est entièrement géré par notre programme. Le slider température s'adapte aux températures de l'ice tea. Il n'y a que deux tailles de sélection lorsque l'ice tea est sélectionné. Le prix est respecté en fonction du format.

d) Extension : Gestion de l'avancement de la préparation

Nous avons fait l'hypothèse que chaque étape de la préparation augmente la barre de progression d'un pourcentage fixe. Ainsi, on divise 100 par le nombre d'étapes pour une recette plus les options.

$$avancement\ (\%)\ =\ 100/(nb\ étape\ +\ options)$$

Lors de chaque étape, on vient compléter le progress bar de ce pourcentage. Toute la barre de progression est gérée au sein du code java car cela ne fait pas partie du contrôle de la FSM.

e) Extension : Détection de gobelets

Durant la phase de sélection à tout moment on peut ajouter sa tasse personnelle qui permettra une réduction de 10 centimes pour la bonne action d'économiser du plastique. Durant la phase de réalisation de la boisson, on ne vient pas ajouter le gobelet. Il est possible de retirer son gobelet personnel. L'affichage s'adapte également au gobelet choisi.

II) Explications de la structure de la state machine

Etape 1 (cf. annexe 1)

Le point d'entrée de notre state machine est *Init.* A chaque fin de préparation on passe dans cet état qui remet à 0 l'interface de sélection et on nettoie toute la machine.

Une fois cet état réalisé on entre dans la phase de sélection et paiement. Cet état de la DrinkFactory est représenté par une parallèle state.

- *Payment* : région qui permet de sélectionner le mode de paiement. Soit on rentre dans *NFC*, soit on insère des pièces et on arrive dans *Coins*. Tant qu'il n'y a pas assez de pièces, on boucle sur *Coins*. Une fois que le paiement est enregistré (CB ou en pièce) on vérifie si le produit sélectionné est disponible. Si oui, c'est à cet endroit que l'on finit le processus de la *UserSelection*. Si la sélection est faite et que le montant est prépayé on sort de *UserSelection*.
- *produceSelection* : région qui permet de récupérer les valeurs des différents sliders. A chaque changement de position, on récupère la valeur tant qu'on est dans le UserSelection.
- *sliderType* : permet de changer les sliders en fonction de la sélection. On met à jour l'interface utilisateur.
- *time* : c'est la région qui gère le fait qu'au bout de 45s on sorte de UserSelection. Chaque fois que l'on appuie sur un bouton de cette interface, on remet à 0 le timer.

L'avantage de mettre le breaking point après avoir vérifié si la boisson est sélectionnée et payée est de sortir automatiquement et de passer à la réalisation mais la limitation vient lorsque l'on a ajouté les options. Ce mécanisme force à sélectionner les options avant le paiement.
Toutes ces régions représentent le contrôle dans la fsm. Elle gère les événements nécessaires pour la réalisation de la sélection et paiement.
La gestion de l'affichage des sliders dans la fsm est questionnable. Les options ne sont pas mises dans la FSM par choix, cela permet une gestion plus simple de ces dernières dans le code java plutôt que dans la FSM.

A tout moment on peut sortir de userSelection et retourner à l'état initial en pressant cancel, ce qui engendre un remboursement.

Lorsque l'on sort de userSelection on arrive dans *Pay* qui réalise le paiement.
Calcul du prix et mémorisation des options choisies : A l'état reset, un objet *Drink* est créé. Il va stocker les valeurs des sliders, le type de boisson, ainsi que les options

(Annexe 5). Il contient une méthode qui va calculer son propre prix, et la retourne à la *DrinkingMachine* lorsqu'elle en a besoin : pour l'afficher, ou pour effectuer le paiement.

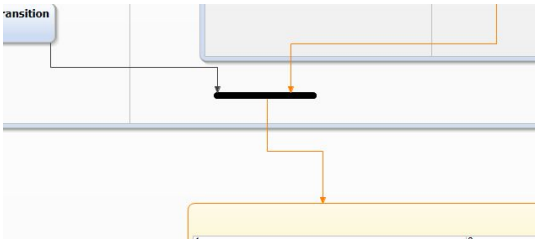
Etape 2 (cf. annexe 2.a)

Nous arrivons ensuite dans la parallèle state firstStep. Cette parallèle state gère le chauffage de l'eau et en même temps la mise en place de la dosette adéquate en fonction de la sélection.(cf annexe 2.b). Les stocks sont décrémentés dans *start* lorsque l'on est sûr que le processus de réalisation est lancé.

Lorsque les dosettes sont placées on entre dans une parallèle state *step2* (cf annexe 2.c), qui nous permet pendant que l'eau chauffe à la température adéquate de venir placer une tasse tout en réalisant l'étape supplémentaire de l'expresso ou de réaliser tout simplement les étapes de la soupe comme indiqué dans la recette. On place la dosette de soupe ainsi que les épices puis une fois réalisé on synchronise pour sortir de cette étape faite en parallèle. L'état transition dans step2 est présent pour continuer le processus et rejoindre la barre de synchronisation. Sans cet état on devrait attendre la fin de réalisation d'étape faisant partie de recettes différentes, ce qui est impossible.

Pour sortir de firstSteps nous voulions faire cela avec une synchronisation, afin d'attendre que l'eau soit chaude et que les étapes de placer la dosette et faire des étapes de recette spécifique soient fini pour passer à l'étape suivante, mais dû à un bug probable de Yakindu, nous avons utilisé une étape de transition à nouveau. Ainsi on boucle dans la fsm sur l'état waterHeat qui incrémente la température de +2° toutes les secondes. Toutes les secondes, on vient faire un contrôle sur cette température grâce à isHot qui compare la température souhaitée et la température actuelle. Lorsque que la température souhaitée est atteinte isHot renvoie true et on peut passer à l'étape suivante et on sort de step1.

solution actuelle



ancienne solution, avec un problème de synchronisation

L'avantage de mettre la parallèle state pour le chauffage de l'eau et ensuite encapsuler une autre parallèle state pour la cup et la recette, est le gain de temps et la puissance de tout faire en même temps. La limitation de cette solution est le fait qu'elle est difficilement extensible dans le cas ou l'on aurait plus de boisson, il faudra probablement ajouter des états et potentiellement changer la façon de faire si cela pose problème.

Etape 3 (cf. annexe 3)

Nous rentrons ensuite dans *LastSteps* qui est la parallèles state qui permet de faire couler l'eau et en même temps, en fonction de la sélection ajouter les croûtons s'ils ont été sélectionné ou ajouter le sucre en quantité désiré. On utilise le même système ici avec un état de transition et un booléen pour attendre que l'eau soit complètement versée dans le gobelet pour contourner le bug.

La sortie pour un expresso ou un café mène à l'état permettant d'ajouter de la crème glacée si cette dernière a été choisie en tant qu'option. Pour les autres sélections, on se dirige directement vers *FinalStep*.

Etape 4 (cf. annexe 4)

On rentre ensuite dans l'état final qui permet de réaliser les dernières étapes des recettes les plus complexes. On laisse infuser le thé et l'ice tea, on laisse refroidir l'ice tea et on rajoute le lait pour le café, thé ou expresso si besoin. Pour le refroidissement de l'ice tea on utilise le même système que pour le chauffage, on boucle sur une condition jusqu'à ce que cette dernière soit vraie. Une fois fini on revient à l'état initial, on nettoie la machine, on réinitialise les paramètres (tout en gardant en mémoire les ingrédients et cartes bleues) et le cycle peut de nouveau s'exécuter.

III) V&V (LTSA)

Nous avons réalisé plusieurs tests. Grâce à LTSA nous avons essayé de reproduire le plus fidèlement possible notre FSM. Nous avons effectué des tests de liveness dans le but de s'assurer du bon fonctionnement de la machine. Par exemple, nous avons fait le chemin d'exécution pour faire un café et on s'est assuré que les étapes étaient bien réalisées, lorsque je sélectionne un café j'ai éventuellement le paiement du café et je peux le récupérer à la fin du processus. De même on a réalisé

le même type de tests sur les options, si je sélectionne du lait on me rajoute bien du lait dans ma boisson.

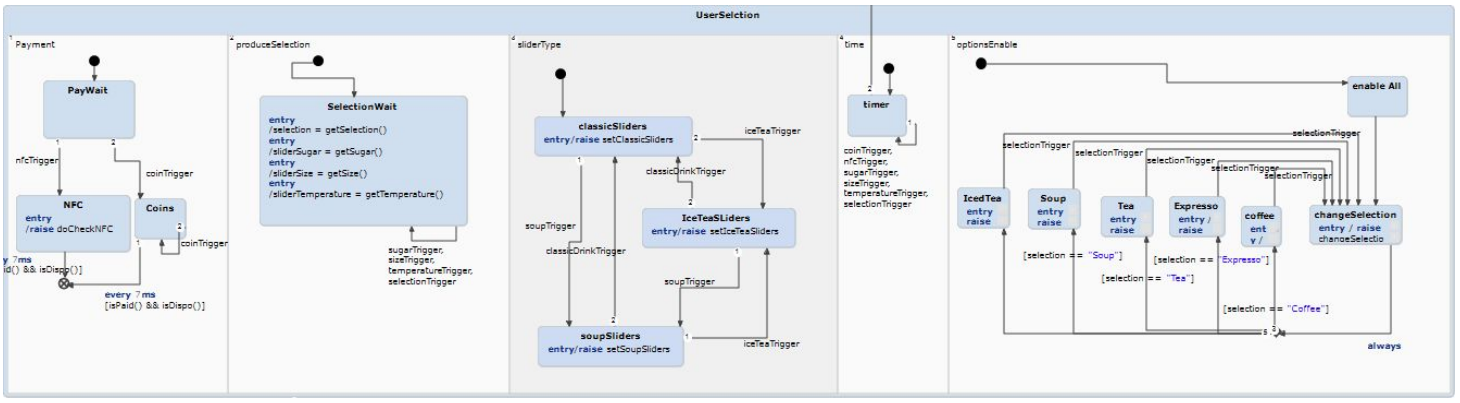
IV) Prise de recul

Nous avons rencontré quelques difficultés concernant l'utilisation de Yakindu et fait face à certains bugs du logiciel, mais nous avons réussi à contourner la plupart de ces problèmes techniques.

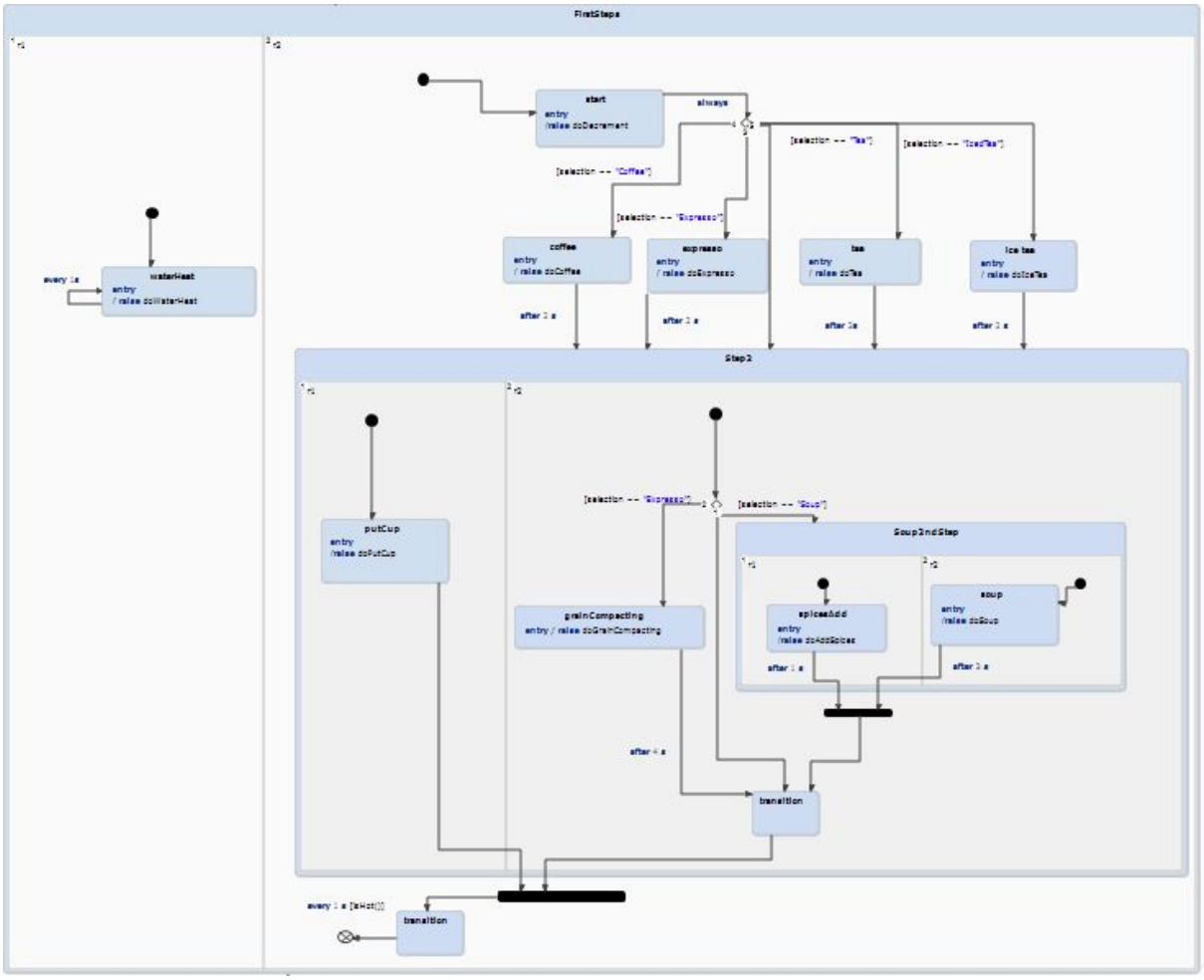
Concernant le code, certaines parties peuvent être réalisées plus agilement. Certaines méthodes java sont très lourdes et pourraient être refactorées. Certaines hypothèse on été prise concernant les spécifications, par exemple le fait de devoir renseigner les options avant de payer si on veut rajouter une option à une boisson dans le cas contraire vous ne pouvez faire que la version basique de la boisson.
De plus l'affichage n'est pas complètement dynamique lors de la sélection des options et on ne spécifie pas qu'une option n'est pas possible lors de la sélection, cette dernière n'est juste pas prise en compte lors de la réalisation de la recette.
Un axe d'amélioration pourrait être le fait de mieux gérer les boissons dans la fsm dans le sens où certaine boisson passe par des étapes avec un tri en fonction de la sélection alors qu'on pourrait les relier directement au états adéquats, ce qui rendrait moins générique la fsm mais plus performante. Un autre axe d'amélioration serait de réussir à garder les paiement par CB et les ingrédients même lorsque l'on éteint la machine.

Toutefois, ce projet évolue chaque semaine depuis le début, autant suite à une prise de recul du groupe qu'à un changement de spécification. La FSM a été refactorée 3 fois durant le projet. Dans un premier temps nous avions autant de parallèles states que d'étape de la recette les uns après les autres. Puis nous avons fait face à des bugs et avons donc eu l'obligation de trouver une solution.
Tout ce processus nous a permis d'améliorer notre capacité d'adaptation. De plus, nous sommes familier avec le domaine des FSM et comment réaliser un schéma comportant le contrôle qu'il faut pour qu'il puisse être explicite tout en faisant le reste dans le code java. Certes ce projet est loin d'être parfait, tout n'est pas respecté mais nous en sommes fiers, l'ensemble des problèmes rencontrés a généré une grande quantité de travail supplémentaire et nous ont fait perdre énormément de temps, mais nous avons réussi à en surmonter la plupart à deux en fournissant les efforts nécessaires.

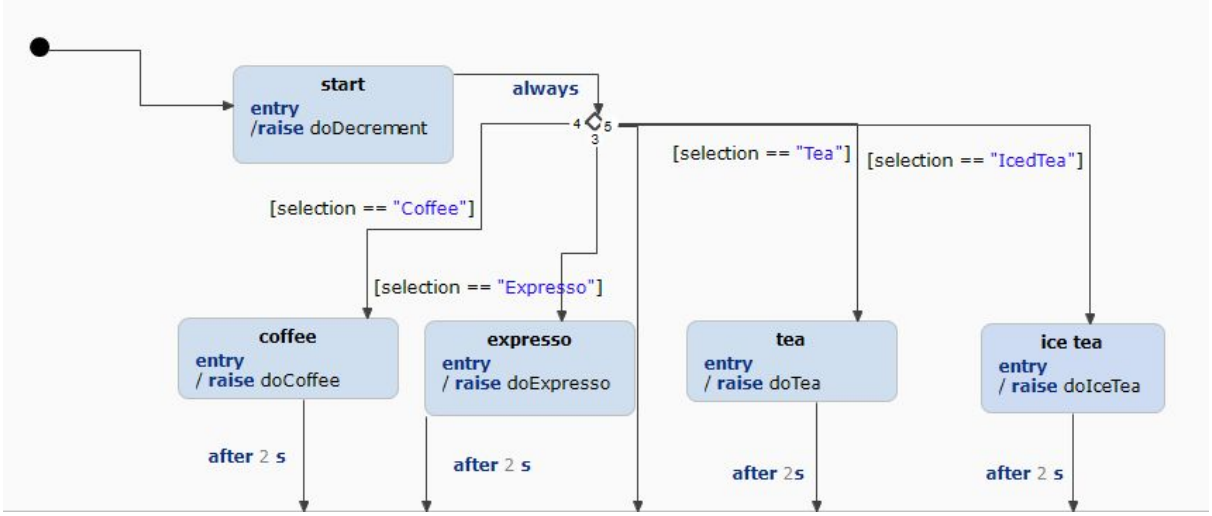
V) Annexes



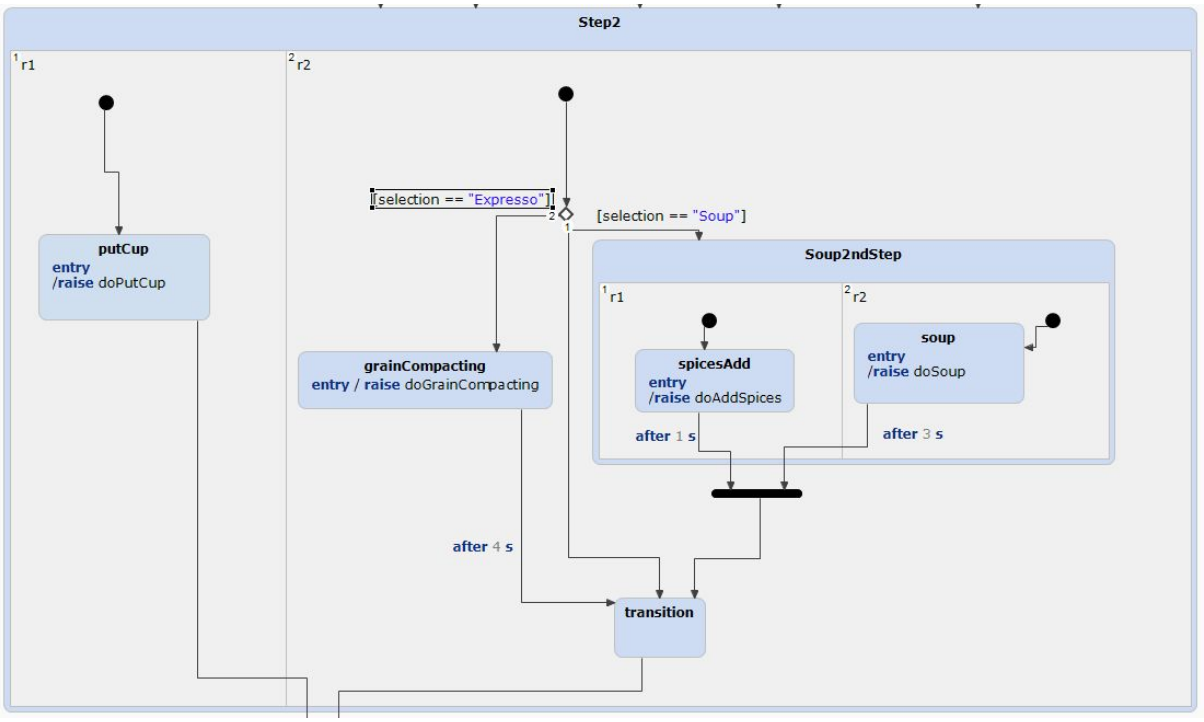
Annexe 1 : UserSelection parallèle state



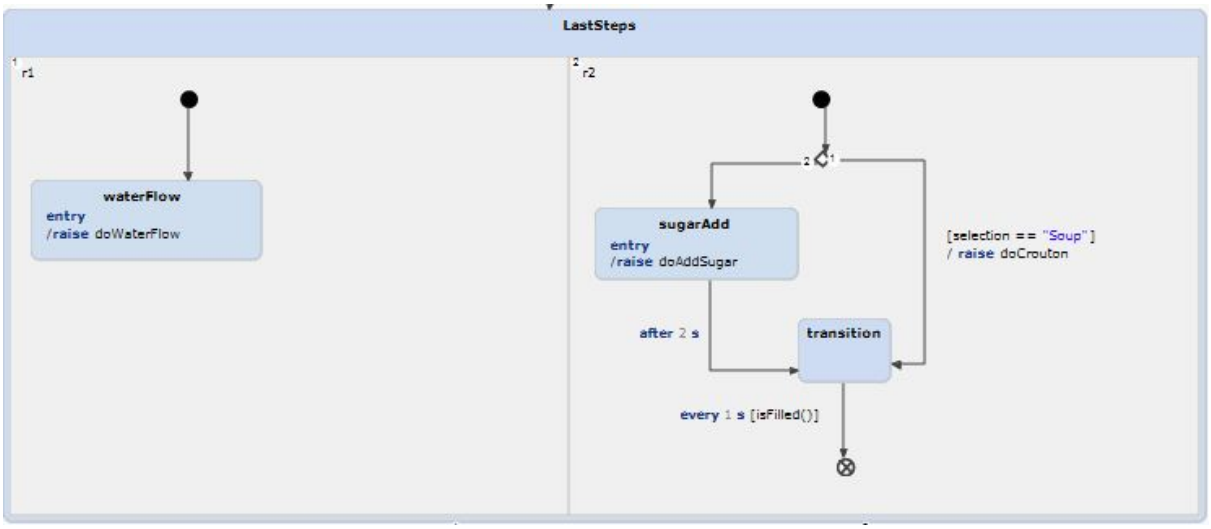
annexe 2.a : FirstSteps parallèle state



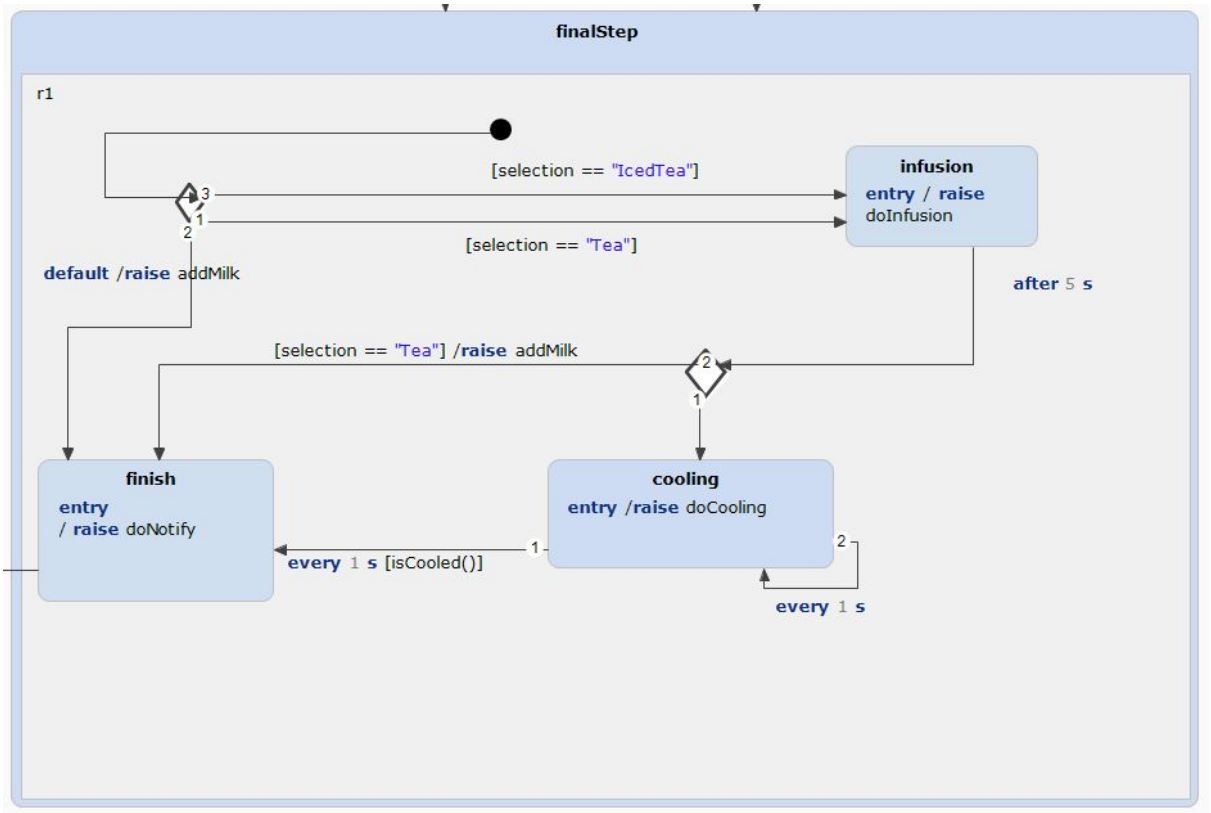
annexe 2.b



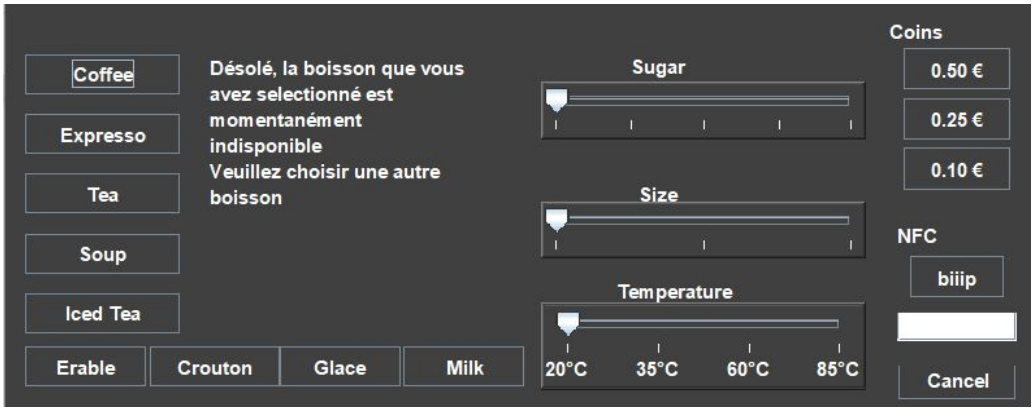
annexe 2.c : Step2



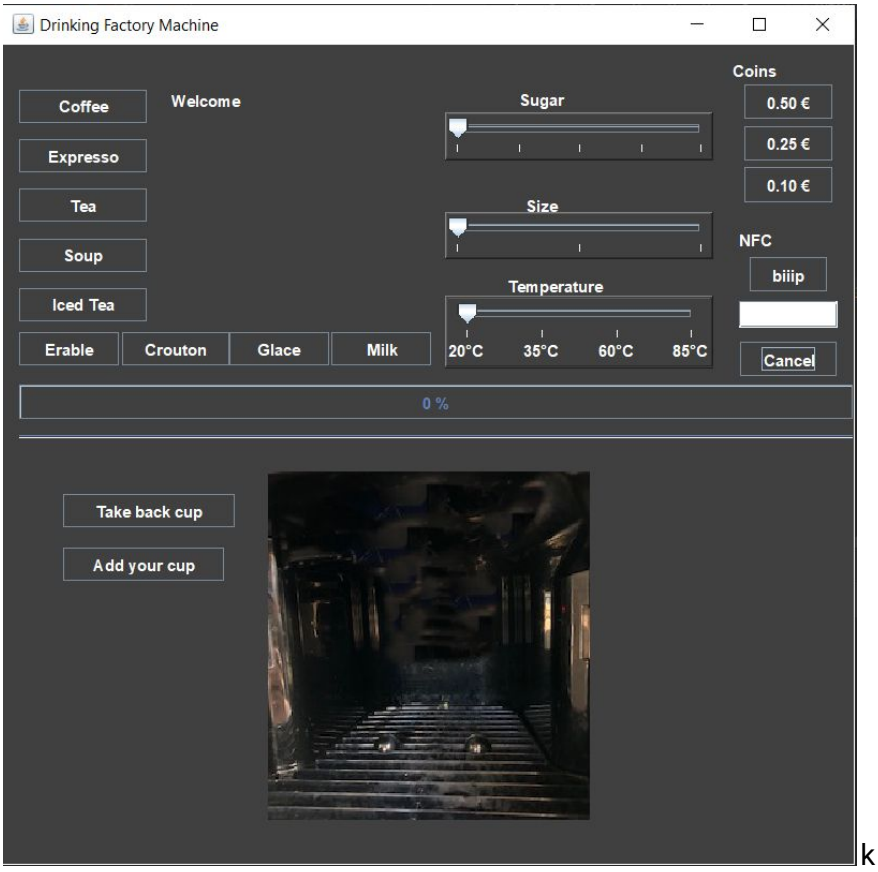
annexe 3 : LastSteps



annexe 4 : finalStep

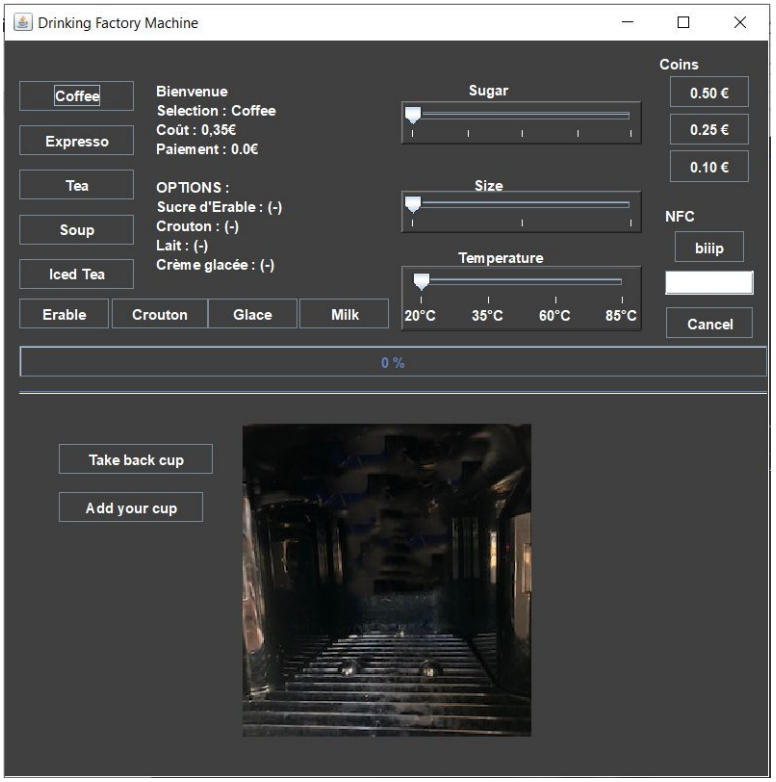


annexe I.a: message rupture de stock

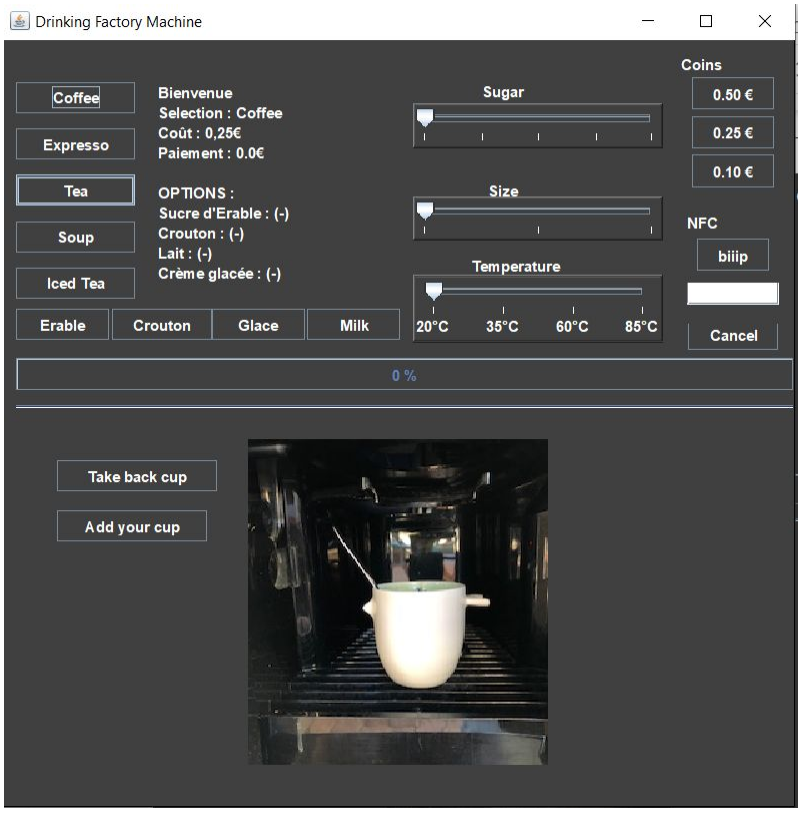


k

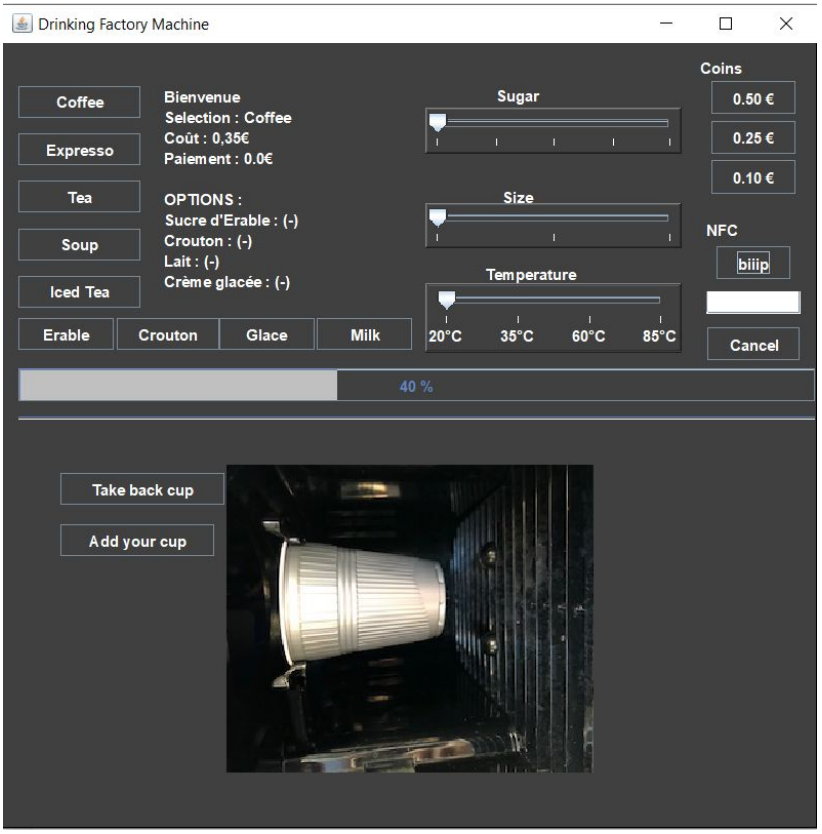
annexe I.b : Aperçu de l’interface utilisateur



Annexe I.c : UI lors de la sélection



Annexe I.d : Ajout tasse personnelle



Annexe I.e : Ajout tasse polluante et vision barre de progression

```
public class Drink {
    enum DrinkType {
        COFFEE,
        EXPRESSO,
        TEA,
        SOUP,
        ICEDTEA
    }

    DrinkType type;
    int CondimentDose;
    int size;
    int temperature;
    boolean erableOption;
    boolean croutonOption;
    boolean glaceOption;
    boolean milkOption;


    public Drink() {

    }
```

Annexe 5 : attributs et constructeur de *Drink*