

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. (33)2-47-36-14-14
Fax (33)2-47-36-14-22
www.polytech.univ-tours.fr

Rapport
Projet de Programmation et Génie Logiciel

Simulation d'un service du midi à la K'Fet

Auteurs

Aurane Martin
[aurane.martin@etu.univ-tours.fr]

Thomas Blumstein
[thomas.blumstein@etu.univ-tours.fr]

Encadrant

Yannick Kergosien
[yannick.kergosien@univ-tours.fr]

Polytech Tours
Département Informatique

Table des matières

Introduction	1
Ressources	2
Spécifications	4
Simulation	6
A. Evénements	6
B. Code	8
1. Générateur	9
2. Simulateur	10
Résultats	12
A. Clients	12
B. Appareils	14
C. Listes d'attente	15
D. Kfetiers	16
E. Affichage Graphique	18
Conclusion	20
Annexes	22
Annexe A : Table des Figures	22
Annexe B : Diagramme d'évènements	22
Annexe C : Diagramme de classes	22

Introduction

Ce projet a pour but de simuler de manière réaliste un service du midi à la K'Fet (meilleure cafétéria de Polytech Tours) de l'arrivée des clients à leur départ avec leurs commandes et ainsi récupérer des données nous permettant d'optimiser nos flux de productions et de service à nos clients.

Notre sujet n'était, à l'origine, pas celui-ci ; en effet nous devions simuler un centre vaccinal sans rendez-vous mais après discussion avec notre tuteur nous avons décidé de changer notre sujet et de simuler un service à la K'Fet. En effet, nous faisons tous les deux parties de l'actuelle équipe de la K'Fet et nous avons estimé que c'était un outil qui pourrait nous servir pour optimiser nos services et ainsi offrir une meilleure expérience à nos clients.

Notre choix d'un sujet de simulation était tout de même intéressé par le fait que nous ne connaissions pas grand-chose à la simulation à événement discret et nous voulions en apprendre plus sur ce sujet qui nous permettra d'avoir un panel de compétences plus large.

Pour la bonne compréhension de nos choix, il est utile de comprendre comment fonctionne la K'Fet. Nos activités sont divisées en 3 services : 2 services aux pauses matinales et post méridiennes durant chacun 15 minutes et un service principal lors de la pause méridienne. Pendant les 2 services de 15 minutes nous servons café, sodas, viennoiseries, barres chocolatées et bonbons et lors de la pause méridienne nous servons les mêmes articles mais surtout différents plats (Pizzas, plats surgelés, ramens et depuis peu des croque-monsieur (ces derniers n'étant pas encore à la carte lors du début de notre projet nous ne les avons pas simulés)).

Nous avons décidé que la partie la plus intéressante à simuler était le service méridien. En effet, c'est celui qui est le plus éprouvant pour les kfetiers de par sa longueur mais également par le fait que nous avons plus d'articles à la carte, ce qui explique que nous sommes généralement par équipe de 4 pendant ces services alors que nous ne sommes qu'en duo aux autres services. C'est également le service le plus "optimisable" et donc le plus rentable à simuler.

Nous avons pour consignes principales de créer une application pouvant simuler ce service, avec différents réglages possibles (nombre de ressources humaines, nombre de clients, caractéristiques des clients...) et une interface graphique.

Partie 1 :

Ressources

La K’Fet dispose de plusieurs articles à la carte : Soda, Café, Chocolat chaud, bonbons, barres chocolatées, viennoiseries, plats Picard, ramen et pizza.

Lorsqu’un client commande un soda, des bonbons, des barres chocolatées ou une viennoiserie, le service est considéré comme “instantanée” : il n’y a rien à préparer, le kfetier va simplement chercher la commande et la sert. Ainsi, ces évènements n’auraient été intéressants à simuler qu’en cas de problème dans la commande - problème de type “il faut changer la commande, nous n’avons plus ce produit en stock” - mais cela impliquait qu’il fallait inclure la gestion des stocks dans le projet. Or, nous avons choisi dès le départ de ne pas l’inclure, car l’objectif du projet est de simuler un service de la K’Fet et non pas sa gestion interne. Ainsi, nous n’avons retenu comme branches d’évènements que les commandes de boisson chaude ou de repas.

Après avoir choisi ces branches d’évènements, nous avons dû déterminer quelles sont les ressources nécessaires à leur réalisation : il paraît assez évident que pour faire des cafés il faut une machine à café, pour cuire des pizzas il faut des fours etc., nous avons donc établi la liste suivante :

- 8 Fours
- 1 Bouilloire
- 1 Machine à café
- 1 Machine à Chocolat Chaud
- 3 Micro-ondes

Mais aussi des ressources humaines :

- Caissier
- Kfetier
- Cuisinier

Pour les ressources matérielles, nous avons fixé des nombres en fonction de ce que nous possédons à la K’Fet : évidemment nous n’avons pas 8 fours, mais il nous paraissait compliqué de simuler uniquement 4 fours pouvant cuire deux pizzas à la fois (maintenant que nous sommes à la fin de la simulation, nous savons que ça n’aurait pas été si compliqué mais cela n’aurait pas apporté grand-chose de plus). La machine à café peut préparer deux cafés en même temps, c’est donc comme s’il y en avait deux, et la bouilloire peut préparer deux ramen lorsque l’eau est chaude.

Pour les Kfetiers cependant, notre seule contrainte est d’être 4 au total et au moins un par rôle, nous avons donc laissé la possibilité de choisir quel rôle se fera en duo.

A partir de cette liste de ressources, nous avons pu établir le diagramme d'évènements suivant (Annexe A) :

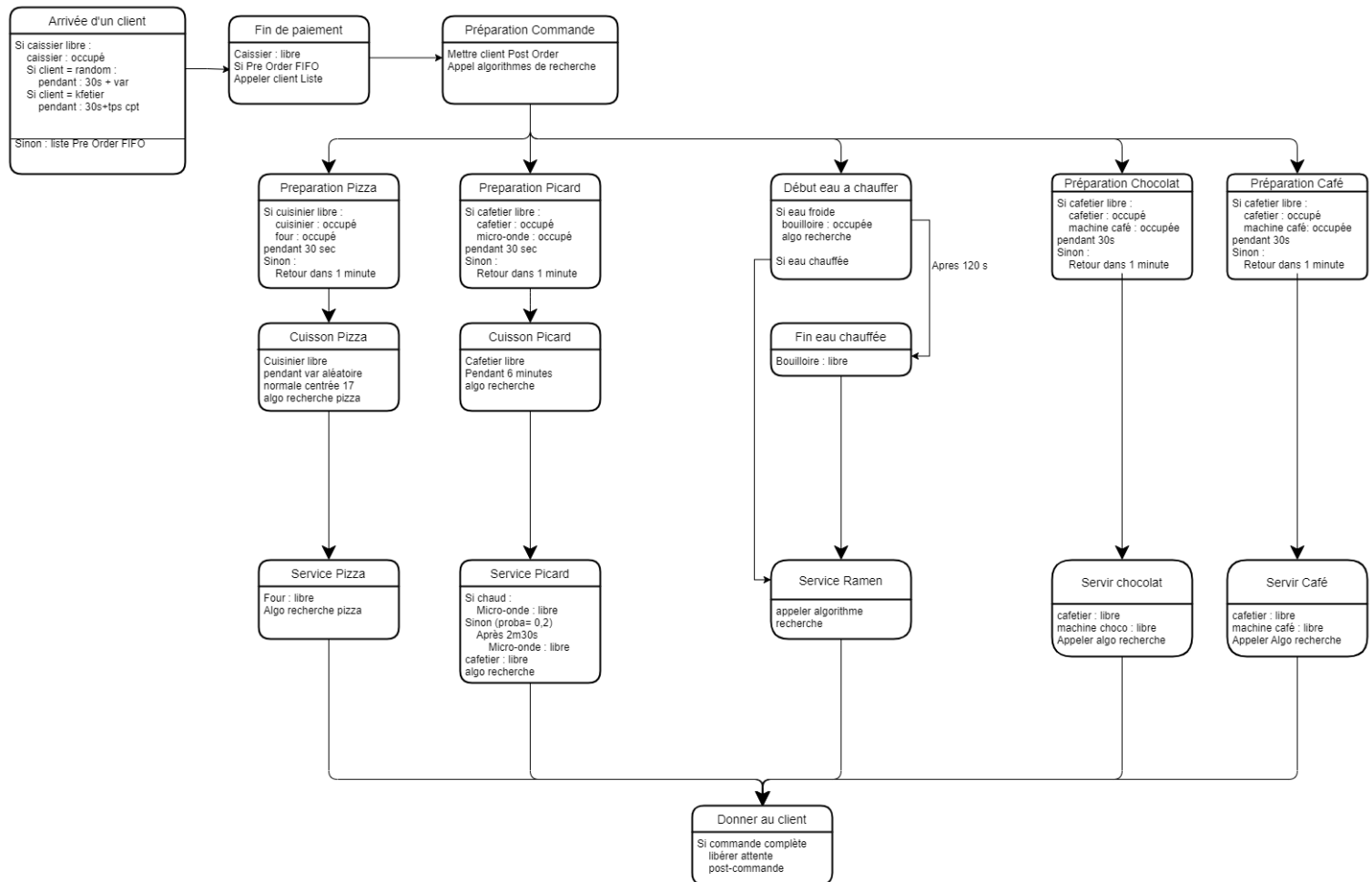


Figure 1: Diagramme d'évènements

Si l'on prend par exemple un client A qui souhaite commander une pizza, un plat picard et un café, pour réaliser sa commande nous aurons besoin d'un emplacement dans la machine à café, d'un four, d'un micro-onde, d'un cuisinier, d'un kftier et d'un caissier. Toutes ces ressources seront gérées par les différents évènements, et chaque évènement ne pourra se lancer que lorsque les ressources nécessaires seront libres.

Partie 2 :

Spécifications

Suite aux consignes données et au diagramme des événements de notre simulation nous avons pu extraire différentes spécifications :

➤ Créer une liste de clients

Chaque client doit avoir une liste d'articles commandés et un temps d'arrivée à la K'Fet. La liste de client sera pré-générée dans un fichier donné au simulateur pour reproduire une simulation avec des paramètres différents.

➤ Paramétrer la simulation

La simulation doit pouvoir être réglée pour simuler différents scénarios. Il doit notamment être possible de régler l'occupation des ressources humaines (4 ressources humaines réparties en 3 postes : caissier, cafetier et cuisinier)

➤ Créer et lancer des événements

La simulation reposant sur une suite d'événements, il faut pouvoir créer ces différents événements pour représenter chaque action des cafetiers (faire cuire une pizza, servir un café, encaisser un client...). Suite à leur création, il faut pouvoir lancer les événements au bon moment pour avoir la simulation la plus réaliste possible.

➤ Exporter les résultats

Les résultats de la simulation doivent pouvoir être analysés à la fin de la simulation. Pour cela, nous allons exporter les données dans un fichier .xlsx qui nous permettra une analyse graphique et différents calculs sur ces données.

➤ Afficher la simulation

La simulation doit être visuelle, il faut donc une interface graphique représentant la simulation et montrant les différentes actions des cafetiers et des appareils.

A l'aide de ces spécifications, nous avons pu concevoir le diagramme des classes de notre application (Annexe B) :

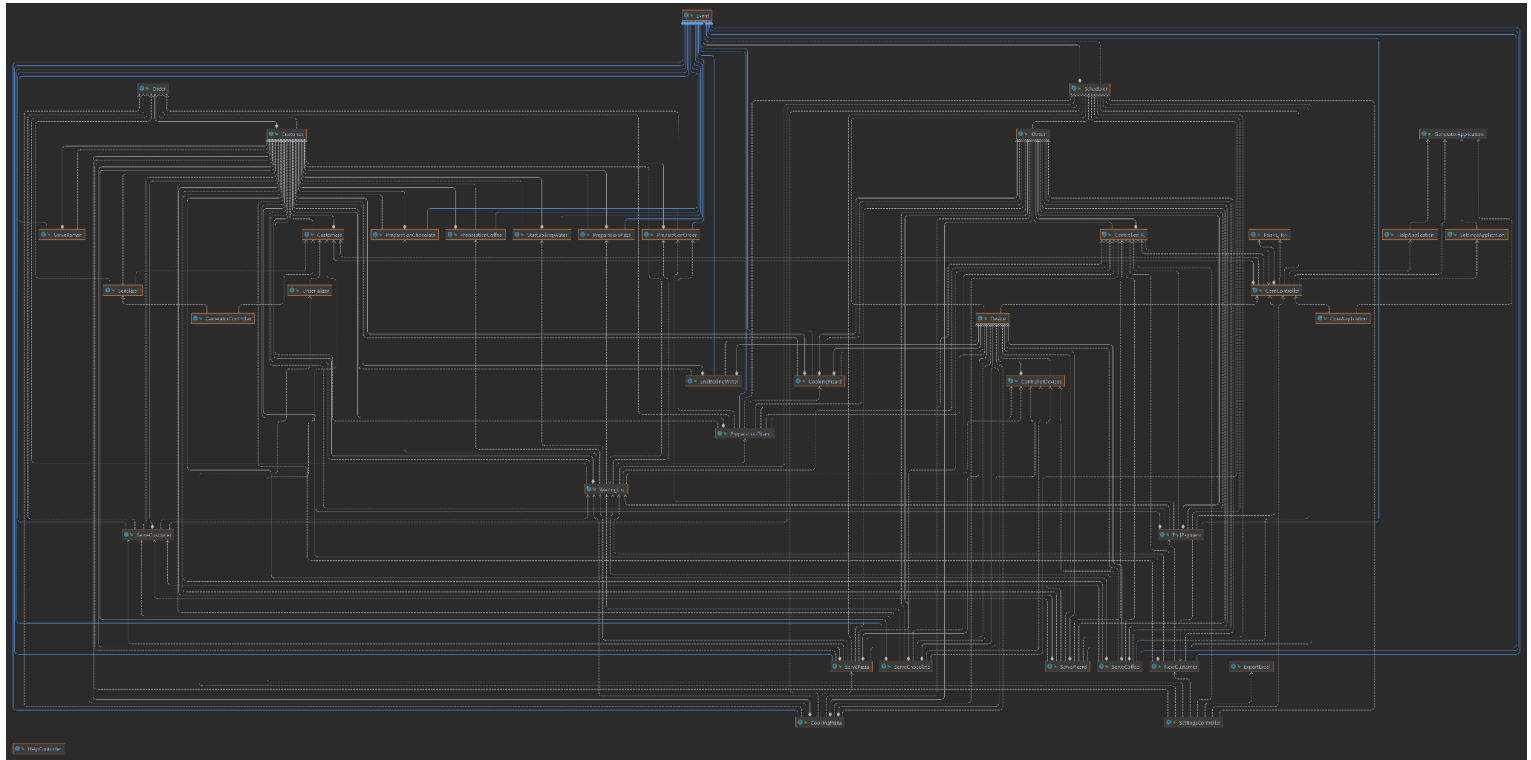


Figure 2: Diagramme de classes

Comme on peut le voir, la hiérarchie de ce projet n'est pas des plus simples. Pour résumer nous avons créé la majorité de nos classes d'après le diagramme des événements. En effet, chaque événement correspond à une classe du programme. A ces classes d'événements nous ajoutons également toutes les classes servant au générateur de clients, au réglage de la simulation, à l'affichage de celle-ci et à l'exportation des données.

Partie 3 :

Simulation

A. Événements

Comme nous pouvons le voir sur la [Figure 1], nous avons au total dans ce projet 17 évènements, ainsi que deux algorithmes de recherche permettant d'articuler les évènements entre eux. Le diagramme se sépare en "branches", représentant ainsi les évènements nécessitant les mêmes ressources, avec par exemple :

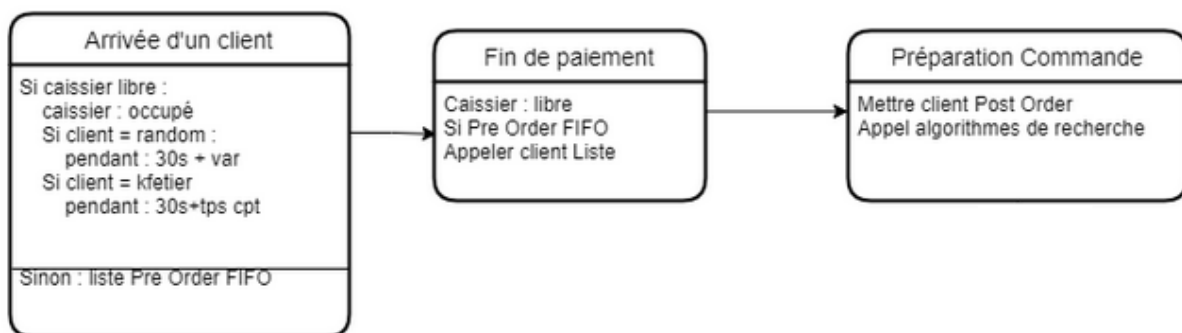


Figure 3: Branche caissiers

Cette branche représente la branche des caissiers : lorsqu'un client arrive, il va falloir vérifier qu'un caissier est bien disponible pour prendre sa commande, si ça n'est pas le cas alors le client sera mis dans une liste d'attente de type FIFO (on ne voudrait pas que les clients se doublent). Si un caissier est disponible, alors il peut commencer à prendre sa commande, et le statut de la ressource caissier passe donc à "occuper" pendant un certain temps (représentant le temps de formuler la commande et d'encaisser le client) selon si le client est un kfetier ou pas. Lorsque le temps est écoulé, l'évènement Fin de paiement est appelé et libère la ressource du caissier. S'il n'y a pas de client dans la liste d'attente, alors le caissier ne fait rien tant qu'un client n'arrive pas, sinon il appelle le client suivant ce qui lance un évènement Arrivée d'un client. Si l'évènement Fin de paiement est lancé à un temps t , alors l'évènement Préparation Commande sera lancé à $t+1$: le client est placé dans la liste d'attente Post Order (celle-ci n'étant pas FIFO : il n'y a pas de raisons que le client n'ayant commandé qu'un café doive attendre que le client ayant commandé une pizza soit servi pour être servi à son tour) et les algorithmes de recherche sont appelés.

En quoi consistent nos algorithmes de recherche ?

Il s'agit d'algorithmes qui vont lancer les différents évènements de préparation de la commande. Nous avons un algorithme pour toutes les préparations de commandes nécessitant une ressource Kfetier, et un autre pour les pizzas (qui nécessitent donc une ressource Cuisinier).

Les algorithmes vont d'abord vérifier qu'il y a au moins un Kfétier/Cuisinier disponible pour lancer la préparation de la commande. Ensuite, ils vont vérifier que les ressources matérielles nécessaires à la préparation de la commande sont disponibles. Pour l'algorithme de recherche pour les pizzas, il va donc vérifier qu'un four est disponible ; pour l'autre algorithme il va falloir vérifier dans un certain ordre si les ressources nécessaires sont libres selon ce qui est présent dans la commande. Concernant l'ordre, nous l'avons fixé par rapport à nos habitudes lorsque nous sommes en service à la K'Fet : d'abord les plats Picard, puis les ramen, puis les cafés et enfin les chocolats chauds.

Reprenons l'exemple du client A qui a commandé un plat Picard, une pizza et un café : l'algorithme de recherche pour les pizzas va chercher si un cuisinier est libre, le cas échéant il cherchera si un four est libre, et si c'est le cas alors l'évènement Préparation pizza sera créé. En parallèle, l'autre algorithme va regarder si un Kfétier est libre, le cas échéant il ira d'abord chercher si un micro-onde est libre (le cas échéant il lancera l'évènement Préparation Picard), puis n'ayant pas de ramen dans la commande il ira directement regarder s'il y a un emplacement de libre dans la machine à café pour créer l'évènement Préparation Café. Chaque évènement de préparation de commande occupe les ressources nécessaires et lance les évènements suivants, et à chaque fois qu'une des ressources humaines nécessaires est libérée, alors les algorithmes de recherche sont rappelés pour poursuivre la préparation de la commande.

La seule branche dérogeant à cette règle est celle de la préparation de Ramen :

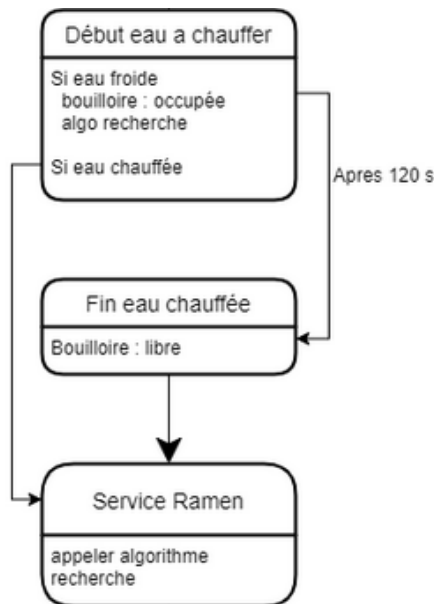


Figure 4: Branche Ramen

En effet, lorsque la bouilloire a chauffé, le service de ramen est immédiat. Mais si l'eau est froide, il faut d'abord attendre que l'eau chauffe avant d'appeler l'évènement Fin eau chauffée pour servir le Ramen, tout en sachant que l'eau refroidit en 10 minutes.

B. Code

Notre application de simulation est en fait constituée de plusieurs applications imbriquées. Pour chacune, nous avons utilisé le modèle Vue-Controller et Java FX afin de créer une interface graphique avec laquelle notre code peut interagir. Nous avons aussi eu besoin de différentes librairies dont java fx et apache.poi pour générer le rapport en fin de simulation. Enfin, nous avons à plusieurs reprises utilisé le design pattern du singleton pour différentes classes, afin de s'assurer que nous n'aurions qu'une seule instance au cours de la simulation : c'est par exemple le cas pour notre Scheduler, ou notre classe WaitingList.

En termes d'application, nous avons donc l'application principale (module Core), ainsi qu'un générateur (module Generator). Pour pouvoir lancer la simulation, il faut d'abord avoir généré une clientèle en passant par le générateur, comme expliqué dans l'aide du logiciel. Il faut ensuite fournir le fichier aux paramètres de la simulation, et seulement une fois le fichier fournis la simulation pourra être lancée.

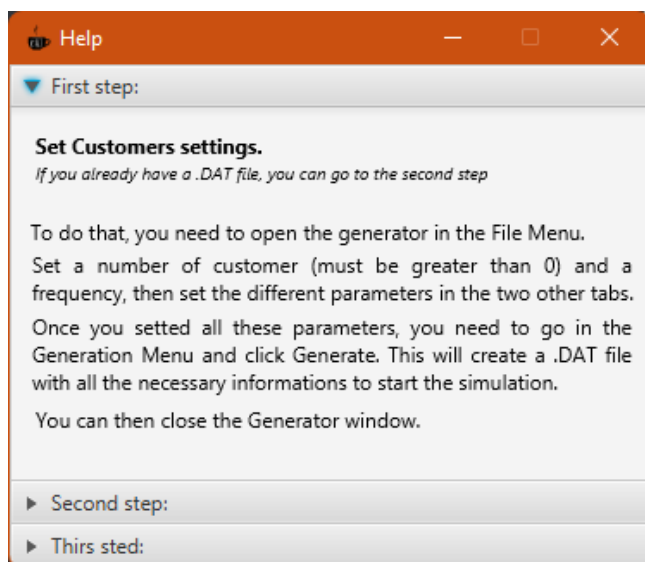


Figure 6: Aide (1)

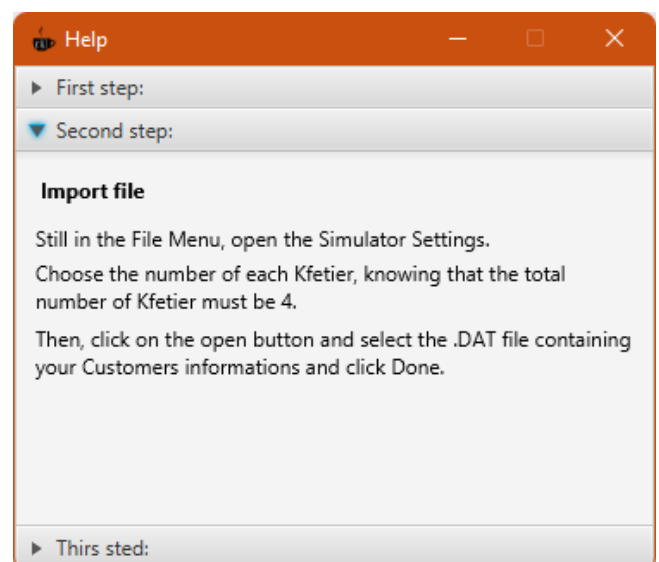


Figure 5: Aide (2)

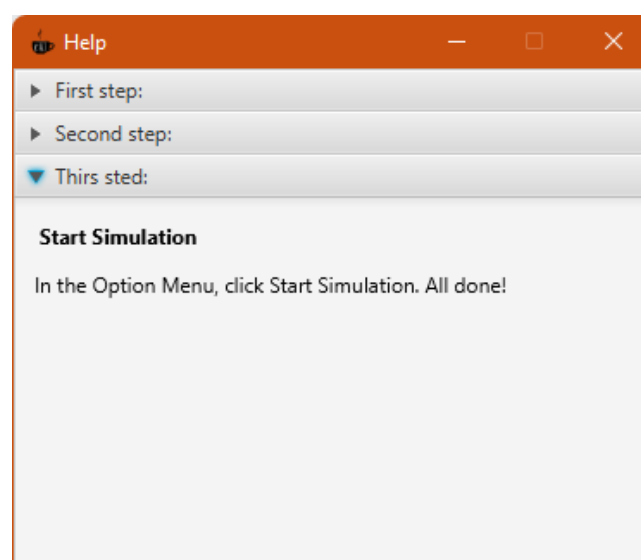


Figure 7: Aide (3)

1. Générateur

Le générateur est composé de 3 onglets, permettant de régler différents paramètres pour la clientèle et ses commandes.

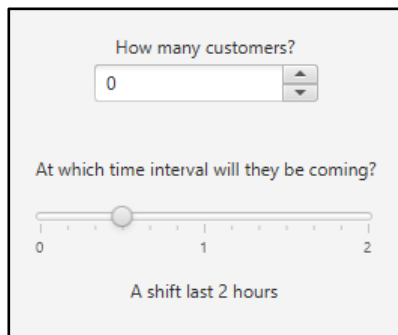


Figure 8: Générateur (1)

La [Figure 7] représente le deuxième onglet du générateur, où il est possible de changer la probabilité qu'un client soit un kfetier (ce qui influe sur son temps de paiement), mais aussi le temps que prend chaque moyen de paiement. Nous avons choisi de ne mettre que 4 moyens de paiements, car c'est ce que nous observions à la K'Fet au début du projet (la carte bancaire n'a été ajoutée comme moyen de paiement que récemment à la K'Fet): il y a donc le paiement par compte qui est réglé sur 3 secondes, celui par Lydia dit "rapide" (le client génère son QR code directement) réglé sur 7 secondes, celui en liquide réglé sur 15 secondes, et enfin celui par Lydia dit "lent" (le client ne trouve pas le QR code, ou sa banque demande une autorisation) réglé sur 20 secondes.

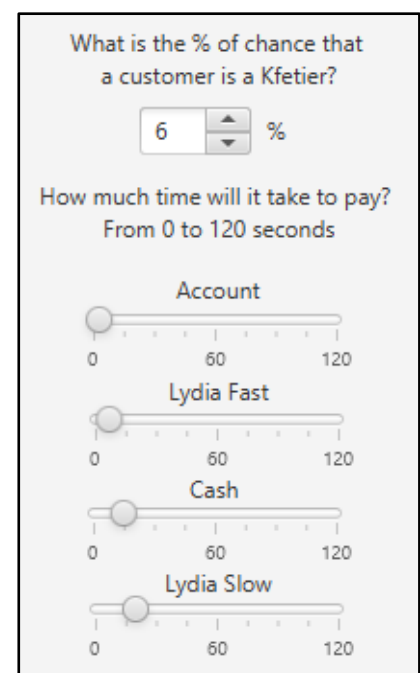


Figure 9: Générateur (2)

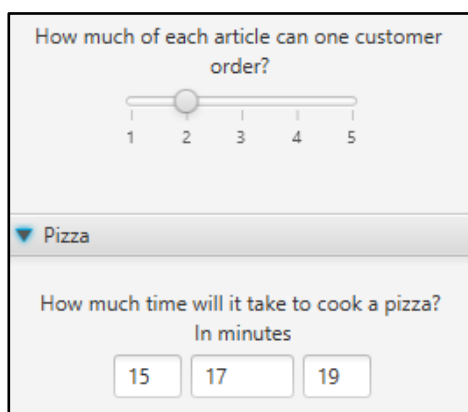


Figure 10: Générateur (3)

Enfin, la [Figure 8] représente le dernier onglet, où il est possible de choisir des paramètres sur les commandes des clients : combien il va pouvoir commander d'article de chaque catégorie, ainsi que le temps que mettra une pizza à cuire (le temps de cuisson est calculé d'après une loi normale centrée sur 17 comprise entre 15 et 19 d'après les valeurs sur la figure).

Lorsque l'utilisateur clique sur générer, les données choisies sont alors sérialisées : la clientèle est alors générée dans un fichier .dat, avec un nombre d'article et une heure d'arrivée aléatoire, et un fichier .txt est créé permettant à l'utilisateur de lire les informations de la clientèle générée.

Ce mode de génération est utile pour pouvoir réaliser plusieurs fois la même simulation tout en gardant des conditions identiques : comme certaines données sont générées aléatoirement, il est nécessaire qu'elles soient stockées quelque part pour pouvoir les réutiliser, c'est donc cette solution que nous avons choisie.

2. Simulateur

Le simulateur est la fenêtre principale de l'application. Il est composé d'une barre de menu et d'un affichage graphique représentant la K'Fet, le timer, les Kfetiers et les appareils ainsi que leur indicateur d'occupation.

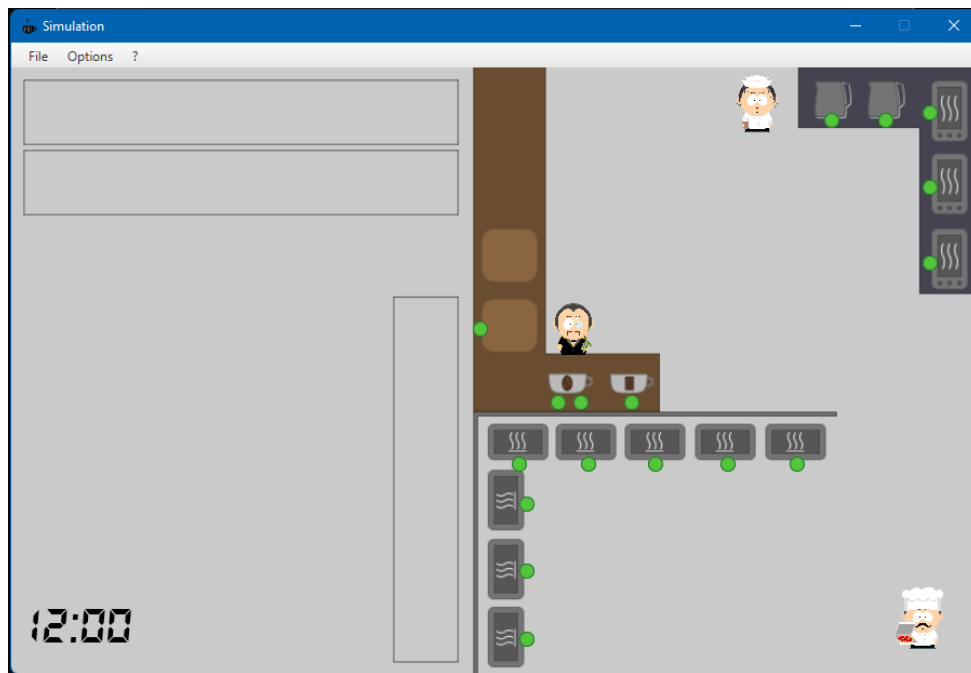


Figure 11: Interface Graphique du simulateur

Dans le menu File, il est possible d'ouvrir le générateur dont il a été question précédemment, et c'est aussi dans ce menu que se trouve la fenêtre de paramètres permettant de transmettre le fichier de clients à l'application. Dans cette fenêtre il sera aussi possible de changer le nombre de Kfetier à chaque post, tout en gardant à l'esprit qu'il y a maximum 4 Kfetier dans un service, et que chaque rôle doit être pourvu. Lorsque le fichier sélectionné est envoyé depuis la fenêtre de paramètres, il est alors désérialisé, et la clientèle générée est alors récupérée dans l'application. Une fois le fichier transmis à l'application, il est possible de contrôler l'application depuis le menu "Options" : on peut lancer la simulation, l'arrêter, la mettre en pause et la reprendre à l'aide des boutons de ce menu.

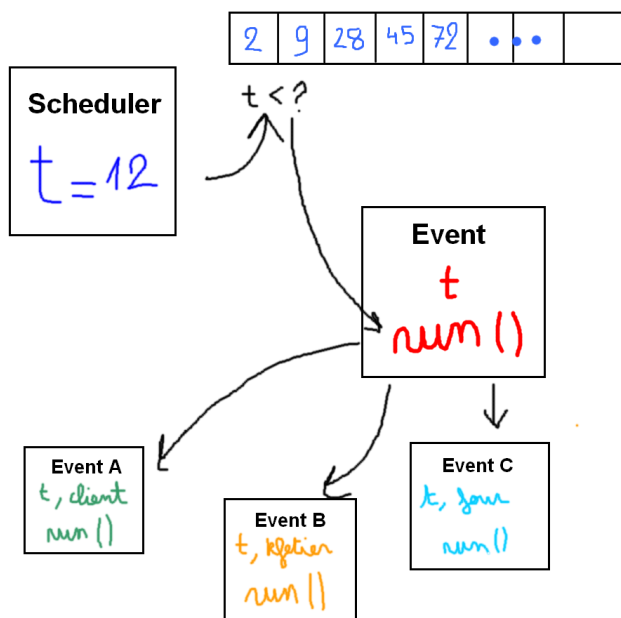
En apparence, la simulation a l'air simple : les ronds changent de couleur et les kfetiers se déplacent, mais derrière ce sont plus de 30 classes qui s'articulent pour parvenir à ce résultat.

Afin de ne pas trop rentrer dans les détails, nous allons simplement expliquer le fonctionnement global de la simulation :

Tout d'abord, l'application est lancée dans un thread pour séparer le côté graphique du côté calculatoire afin de pouvoir effectuer les deux en parallèle.

Ensuite, nous avons une classe Scheduler qui représente l'axe principal de la simulation : c'est dans cette classe que se trouve la méthode qui fait passer le temps dans la simulation. Nous avons choisi d'incrémenter le temps en se basant sur des ticks, en déterminant que 1 seconde dans la simulation serait égale à 0,01 secondes dans la réalité afin que la simulation ne soit pas trop longue mais reste tout de même observable du côté interface graphique.

Nous avons ensuite implémenté une méthode qui, à chaque seconde passée dans la simulation, va rechercher les événements qui doivent être lancés à cette heure précise, ou à une heure antérieure qui n'aurait pas été lancée par manque de ressources disponibles. Pour se faire, tous les événements sont ajoutés à une liste dans un ordre croissant de leur heure de lancement qui est parcourue à chaque seconde jusqu'à ce que l'heure de lancement d'un événement soit supérieure à l'heure actuelle.



Pour implémenter les différents événements, nous avons créé une classe abstraite composée uniquement d'un argument StartingEvent et d'une méthode run(). Nous avons ensuite créé une classe pour chaque événement de la [Figure 1] qui étend cette classe abstraite, et qui est composée d'arguments nécessaires au fonctionnement de l'événement ainsi que de l'implémentation de la méthode run.

Ainsi, pour que chaque événement puisse appeler le suivant, il y a juste à ajouter un nouvel objet de la classe de l'événement en question à la liste d'événement du Scheduler avec une heure de lancement, et le tour est joué.

Figure 12: Schéma du fonctionnement de la simulation

Lorsque le temps est écoulé, la simulation se termine et un rapport est alors généré, afin de pouvoir étudier les résultats de la simulation mais aussi les comparer.

Partie 4 :

Résultats

Le rapport est généré sous la forme d'un fichier Excel composé de 4 feuilles, chacune permettant d'analyser un aspect différent de la simulation. Lorsqu'une simulation est lancée plusieurs fois à partir du même fichier généré, les feuilles de résultats sont ajoutées au même Excel afin de pouvoir les comparer plus facilement. Pour générer ce rapport sous Excel, nous avons utilisé la librairie apache.poi que nous avons importé depuis le repository Maven, qui nous permet donc de créer un fichier Excel, d'y créer des tables et des graphiques ainsi que d'y insérer nos données.

A. Clients

La première feuille permet d'analyser les résultats sur les clients de la simulation, on y retrouve donc :

- Une table contenant
 - L'ensemble des clients de la clientèle
 - Leur heure d'arrivée
 - Leur heure de départ
 - Le nombre d'articles qu'ils avaient commandé
 - Leur temps d'attente total
- Le temps d'attente moyen sur la simulation
- Le temps d'attente minimal
- Le temps d'attente maximal
- Le nombre de clients non servis
- Un graphique affichant le temps d'attente selon le nombre d'articles commandés

Numéro du client	Heure d'arrivée	Nombre d'articles commandés	Heure de Départ	Temps d'attente total
0	61	2	1961	1900
1	65	1	408	343
2	58	3	2111	2053
3	19	3	1540	1521
4	10	5	1140	1130
5	3	5	1389	1386
6	60	2	1372	1312
7	82	4	1508	1426
8	25	1	212	187
9	47	3	1900	1853
			Temps d'attente moyen:	1311
Clients non servis:	0			
			Temps d'attente min:	187
			Temps d'attente max:	2053

Figure 13: Table Clients

Sur cet exemple à 10 clients, on peut par exemple observer que le temps d'attente moyen lors de la simulation était de 1311 secondes, soit environ 21 minutes. Sachant qu'une pizza met entre 15 et 19 minutes à cuire, ce temps est cohérent si sur les 10 clients, plusieurs clients ont commandé une pizza avec d'autres articles. En regardant dans le fichier créé par le générateur, on peut voir que c'est le cas :

Client 2: arrive à 58	Client 3: arrive à 19
Commande: 1 picard	Commande: 1 picard
1 pizza	1 pizza
1 ramen	0 ramen
0 café	0 café
0 chocolat	1 chocolat
Client 4: arrive à 10	Client 5: arrive à 3
Commande: 1 picard	Commande: 1 picard
1 pizza	1 pizza
1 ramen	1 ramen
1 café	1 café
1 chocolat	1 chocolat

Figure 14: Extrait du fichier .txt

Au niveau du graphique obtenu, on peut voir que le temps d'attente selon le nombre d'article commandé reste relativement proche dans une même catégorie. Si l'on regarde pour 2 articles commandés, l'écart entre les deux valeurs peut s'expliquer par la différence d'articles commandés : le point à 1900 correspond à un client ayant commandé un picard et un ramen, tandis que le point à 1400 correspond à un client ayant commandé une pizza et un chocolat. La pizza et le chocolat n'utilisant pas le même kftier, ils ont pu être réalisés en parallèle, réduisant le temps d'attente.

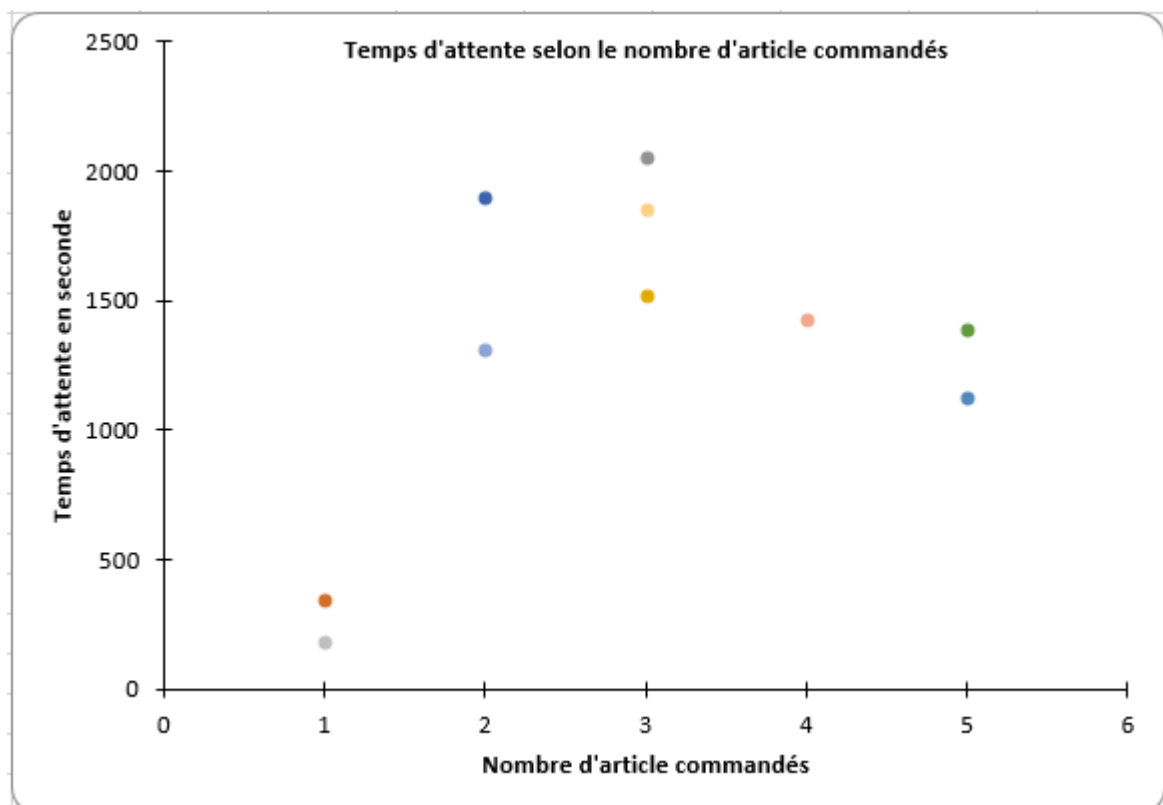


Figure 15: Graphique Clients

B. Appareils

La seconde feuille permet d'analyser les résultats sur les appareils de la simulation. On y retrouve donc :

- Une table contenant
 - La liste des appareils
 - Leur nombre d'utilisation
 - Leur taux d'occupation

Device	Taux d'occupation (%)	Nombre d'utilisations
Micro-Onde n°1	12,9	2
Micro-Onde n°2	12,9	2
Micro-Onde n°3	10,8	2
Four n°1	18,8	1
Four n°2	14,6	1
Four n°3	14,6	1
Four n°4	14,6	1
Four n°5	12,5	1
Four n°6	14,6	1
Four n°7	14,6	1
Four n°8	0	0
Bouilloire n°1	5	2
Bouilloire n°2	5	2
Cafetière n°1	2,1	5
Cafetière n°2	0	0
Machine à chocolat n°1	2,1	5

Figure 16: Table Appareils

Sur 10 clients, 7 ont commandé une pizza, donc seulement 7 fours sont utilisés. Pour la cafetière inutilisée, cela reste cohérent : seulement 5 cafés servis sur un service de deux heures, il suffit qu'ils soient commandés avec seulement cinq minutes d'écart chacun pour que la première machine à café soit libre à chaque fois lorsqu'il y en a eu besoin.

On peut remarquer qu'un des trois micro-ondes à un taux d'occupation inférieur aux autres pour le même nombre d'utilisation, mais cela s'explique très facilement : si pour les autres micro-ondes les deux utilisations correspondent à deux plats picard chauffés, pour celui-ci cela correspond à un plat chauffé qui a ensuite dû retourner chauffer car froid, ce n'est donc pas le même temps de chauffe donc le taux d'occupation change.

C. Listes d'attente

La troisième feuille permet d'analyser les résultats sur les listes d'attente de la simulation. On y retrouve donc :

- Une table contenant
 - La taille de la liste d'attente pré Order (pour passer à la caisse) toutes les 10 secondes
 - La taille de la liste d'attente post Order (en attente de la commande) toutes les 10 secondes
- La taille moyenne de la liste d'attente en caisse
- La taille moyenne de la liste d'attente pour la commande
- Un graphique représentant l'évolution des tailles des deux listes d'attente dans le temps

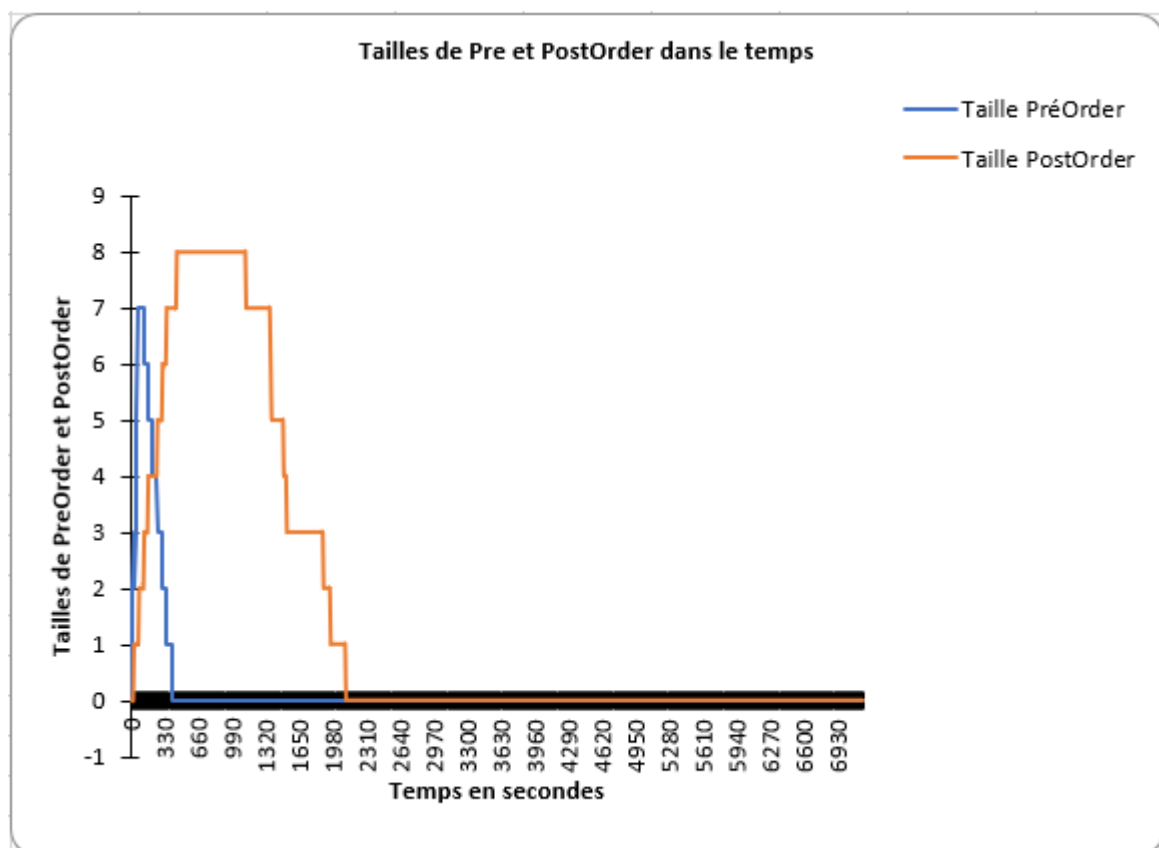


Figure 17: Graphique listes d'attente

Le graphique est la partie de cette feuille qui va le plus nous intéresser : en plus de transmettre les résultats visuellement (les rendant plus faciles à analyser), il permet de comparer les deux ensembles de données rapidement.

En reprenant l'exemple précédent, on peut voir directement que le service n'a en réalité duré que 40 minutes : en effet lorsque la liste d'attente Post Order est vide, c'est que tous les clients ont été servis.

Cela nous permet aussi d'estimer les périodes d'occupation des Kfetier : on peut clairement voir que le début de service est un rush pour le caissier puisqu'il doit s'occuper de tous les clients de la liste Pré-Order à la suite, mais qu'après il n'a plus rien à faire. En réalité, l'arrivée des clients est beaucoup plus espacée et le caissier doit donc travailler plusieurs fois dans le service mais pas en continu. C'est quelque chose qui peut par exemple être simulé en augmentant l'écart entre l'arrivée de deux clients dans le générateur.

D. Kfetiers

Enfin, la quatrième feuille permet d'analyser les résultats sur les Kfetiers de la simulation. On y retrouve donc :

- Une table contenant
 - Les différents cuisiniers
 - Leur taux d'occupation
 - Leur nombre d'intervention
- Une table contenant
 - Les différents caissiers
 - Leur taux d'occupation
 - Leur nombre d'intervention
- Une table contenant
 - Les différents cuisiniers
 - Leur taux d'occupation
 - Leur nombre d'intervention

Avec les feuilles précédentes, il est déjà possible de voir quelle configuration rendra le service plus rapide (est-ce qu'avoir deux caissiers rend le service plus rapide que d'avoir deux kfetiers ?), mais cette feuille permet surtout de voir l'utilité de chaque kfetier : si la simulation est plus rapide avec deux caissiers, cela ne signifie pas forcément que les deux caissiers ont été autant sollicités l'un que l'autre. Avoir ces trois tables permet donc non seulement de comparer les rôles entre eux, mais aussi de comparer les kfetiers au sein d'un même rôle.

Kfetier	Taux d'occupation (%)	Nombre d'interventions
Cook n°1	2,9	14

Figure 18: Table cuisinier

Pour la table du cuisinier, les résultats sont cohérents, il est intervenu 14 fois sachant qu'il y avait 7 commandes de pizzas, il a donc bien mis toutes les pizzas à cuire et sorti toutes les pizzas du four.

Kfetier	Taux d'occupation (%)	Nombre d'interventions
Cashier n°1	6,2	10

Figure 19: Table caissier

Pour la table du caissier, encore une fois tout est cohérent, 10 interventions pour 10 clients à encaisser.

Kfetier	Taux d'occupation (%)	Nombre d'interventions
Kfetier n°1	4,6	22
Kfetier n°2	2,1	6

Figure 20: Table Kfetièrs

Enfin pour la table des kfetièrs, il est plus compliqué de savoir si les résultats sont bons puisque ce rôle doit gérer plus d'évènements différents.

Si l'on compare les trois tables, on peut par contre faire une comparaison intéressante : le cuisinier n'a qu'un taux d'occupation de 2.9%, ce qui est plus bas que pour les autres rôles. On peut donc en déduire que lors d'un service, le cuisinier n'est pas tant sollicité que ça, ce qui est logique puisqu'il n'a qu'à mettre la pizza au four et la sortir 20 minutes après lorsque qu'elle est cuite. Un changement possible pour améliorer les services serait alors de le répartir aux autres postes lorsqu'il n'a rien à faire plutôt que de le laisser uniquement aux pizzas.

E. Affichage Graphique

Notre affichage graphique permet de connaître l'état de la simulation en temps réel. Pour cela nous disposons de plusieurs indicateurs sur l'interface graphique :

➤ Indicateurs d'occupation

Pour nos ressources statiques (ressources matérielles et caissier restant derrière son comptoir) nous utilisons des indicateurs d'occupation sous forme de disques vert et rouge (vert : ressource disponible, rouge : ressource occupée)



Cet indicateur nous permet de facilement voir l'état d'une ressource et de savoir quelles ressources nous pouvons utiliser à un moment donné.

Figure 21: Occupé/Libre

➤ Déplacement des ressources humaines

Nous faisons également se déplacer les différentes ressources "mobiles", c'est-à-dire les cafetiers et cuisiniers se déplaçant entre les différents fours, micro-ondes et autres cafetières. Cela nous permet de savoir où les différentes ressources humaines sont occupées (et accessoirement que certains cafetiers se déplacent beaucoup plus que d'autres)



Figure 22: Cafetiers se déplaçant

➤ Horloge

Pour savoir à quel moment du service nous nous situons, nous avons rajouté une horloge à notre affichage graphique. Cela nous permet notamment de savoir combien de temps il reste et également de voir que la simulation fonctionne quand tous les indicateurs ne changent pas (par exemple quand les fours sont remplis et que nous attendons la fin de leur cuisson)

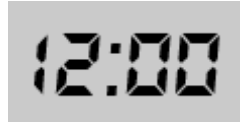


Figure 23 : Horloge

Tous les résultats que nous obtenons grâce à la simulation (que ce soit au niveau de l'affichage graphique ou des données exportées sous Excel) semblent cohérentes avec notre expérience de cafetier. Par exemple, simulation réussit à servir une cinquantaine de client, mais si on augmente le nombre de clients, le point de rupture arrive rapidement. Cela correspond à certains de nos services où nous avons réussi à servir autant de client mais avec difficulté et si nous avions eu plus de client nous n'aurions probablement pas réussi à servir tout le monde.

Conclusion

Ce projet nous a permis d'apprendre le fonctionnement d'une simulation à événements discrets ainsi que la mise en place, les subtilités et difficultés d'un tel programme.

Nous avons également appris, à nos dépends, que c'est un projet plus "lourd" que ce à quoi nous nous attendions et que nous aurions probablement dû garder le sujet original (simulation d'un centre vaccinal sans rendez-vous). En effet, nous voyons plusieurs simplifications dans la simulation du centre vaccinal. Nous pouvons par exemple estimer que les personnes arrivant dans un centre vaccinal vont toutes se faire vacciner et ne vont pas commander pour certains un vaccin et pour d'autre un comprimé pour un mal de gorge, nous aurons donc un seul "chemin" d'événements alors que nous en avons 5 à la K'Fet selon ce que la personne commandera en arrivant (une personne pouvant d'ailleurs prendre différents chemins). Nous avons donc mal estimé le temps que ce projet nous a pris et avons dû prendre des choix pour simplifier notre simulation et ne pas nous perdre dans nos évènements.

Notre affichage a également posé problème. En effet, dès le début nous sommes partis sur un affichage dynamique et très visuel avec le déplacement des différents kfetiers et des indicateurs d'occupation sur nos ressources matérielles. Cet affichage nous a demandé beaucoup de temps de travail et de design que nous n'avons pas pu consacrer au moteur de simulation.

Si nous avions plus de temps à consacrer à ce projet, nous aimerions corriger les problèmes d'affichages graphiques restants. En effet, notre affichage graphique repose sur la modification d'indicateurs d'occupation et de transitions d'images. Mais lors de l'exécution de 2 ou plus de ces actions en simultanée, le programme ne réagit pas comme nous l'aurions voulu et n'exécute pas correctement ces actions ce qui résulte parfois en des comportements incohérents (déplacement incomplet, indicateur d'occupation ne changeant pas d'état).

Ces problèmes viennent de limitations de la part de Java FX, notre bibliothèque graphique, celle-ci a en effet besoin de s'exécuter dans un unique thread pour toutes ses actions et ne peut donc pas effectuer plusieurs opérations en simultanée. Avec plus de temps nous pourrions essayer de multithreader cet affichage pour résoudre les erreurs.

Ce problème nous empêche également pour le moment de simuler l'affichage et le déplacement des clients. En effet, après divers tests nous avons conclu que rajouter un grand nombre d'entités se déplaçant simultanément à nos images de cafetiers ne feraient qu'apporter des comportements incohérents supplémentaires.

Nous aimerions également exporter différentes données statistiques sur le fonctionnement de la K'Fet pour améliorer encore nos services.

Néanmoins, nous sommes fiers d'avoir réalisé ce programme qui nous permettra de mieux structurer et organiser le travail de la future équipe de la K'Fet et ainsi de servir plus d'étudiants affamés tous les midis.

Nous tenons également à remercier notre tuteur, monsieur Yannick Kergosien, pour le temps qu'il nous a consacré, pour nous avoir appris le fonctionnement d'un moteur de simulation et pour avoir répondu à nos différentes interrogations.

Annexes

Annexe A : Table des Figures

Figure 1: Diagramme d'évènements	3
Figure 2: Diagramme de classes	5
Figure 3: Branche caissiers	6
Figure 4: Branche Ramen	7
Figure 5: Aide (2)	8
Figure 6: Aide (1)	8
Figure 7: Aide (3)	8
Figure 8: Générateur (1)	9
Figure 9: Générateur (2)	9
Figure 10: Générateur (3)	9
Figure 11: Interface Graphique du simulateur	10
Figure 12: Schéma du fonctionnement de la simulation	11
Figure 13: Table Clients	12
Figure 14: Extrait du fichier .txt	13
Figure 15: Graphique Clients	13
Figure 16: Table Appareils	14
Figure 17: Graphique listes d'attente	15
Figure 18: Table cuisinier	17
Figure 19: Table caissier	17
Figure 20: Table Kfetiers	17
Figure 21: Occupé/Libre	18
Figure 22: Kfetiers se déplaçant	18
Figure 23 : Horloge	19

Annexe B : Diagramme d'évènements

[Lien de consultation](#)

Annexe C : Diagramme de classes

[Lien de consultation](#)

Simulation d'un service du midi à la K'Fet

Rapport : Projet de Programmation et Génie Logiciel

Résumé

Notre projet consistait en une simulation à événements discrets d'un service du midi à la K'Fet. Ce programme nous permet de simuler le service de l'arrivée du premier client au départ du dernier avec sa commande pouvant être composée de pizzas, ramen, plats surgelés, ou boissons chaudes. Elle comporte un affichage graphique modifié en temps réel pour voir le détail des actions des kfetiers et de l'export des données en fin de simulation vers un fichier Excel pour voir et manipuler ces données notamment sous la forme de graphiques.

Mots clés

K'Fet, kfetiers, simulation, événements discrets, pizza, café, JAVA, Excel, IHM

Abstract

Our project consisted in a discrete event simulation of a lunch service at K'Fet. This program allows us to simulate the service from the arrival of the first customer to the departure of the last one with his order which can be composed of pizzas, ramen, frozen dishes, or hot drinks. It includes a graphic display modified in real time to see the details of the actions of the kfetiers and the export of the data at the end of the simulation to an excel file to see and manipulate these data in particular in the form of graphs.

Keywords

K'Fet, kfetiers, simulation, discrete events, pizza, coffee, JAVA, Excel, GUI

Auteurs/Authors

Aurane Martin

[aurane.martin@etu.univ-tours.fr]

Thomas Blumstein

[thomas.blumstein@etu.univ-tours.fr]

Encadrant/Supervisor

Yannick Kergosien

[yannick.kergosien@univ-tours.fr]

**Polytech Tours
Département Informatique**