# RAPPORT DE PROJET

## Sommaire :



## I. Préambule :

Nous avons passé pas mal de temps pour réfléchir aux différentes subtilités du sujet, en posant bcp de questions et cherchant une structure de code optimale afin de ne pas avoir à revenir en arrière au fur et à mesure de l'avancée du projet. C'est une des principales causes qui expliquent que nous n'avons finalement qu'une seule stratégie concernant la gestion de l'entrepôt. Pour nous, la réflexion et la discussion étaient fondamentales à la réussite de ce projet. Le résultat final allie vitesse d'exécution et optimisation des gains de l'entrepôt. Nous vous expliquons ci-après le déroulement de ce projet et les différents choix, à la fois dans la structure du code et dans les stratégies retenues, que nous avons pris.

**NB :** il n'est pas utile de regarder nos classes tests afin de vérifier le bon déroulement de l'exécution des consignes. Nous vous proposons pour cela la possibilité après l'exécution des consignes (à travers n'importe quel mode d'interaction) de créer un fichier 'sortie.txt' qui vous montrera l'évolution de l'entrepôt, du personnel, etc… à chaque pas de temps, vous permettant de comprendre et de suivre avec facilité le suivi de notre programme.

## II. Répartition du travail :

Pour réaliser notre projet nous avons d'abord comme dit précédemment discuté conjointement et essayé de bien comprendre le sujet. Nous avons ensuite imaginé l'architecture qu'aurait notre programme (les types de classes : interface, hérédité, abstraite ; mais aussi si l'ajout de classes supplémentaires était nécessaire au bon fonctionnement de notre projet). De ce fait, nous avions commencé par créer 2 classes filles (ChefStock et ChefBrico) à la classe Chef, mais nous nous sommes rendu compte que le chef stock n'est pas utile car il n'a pas d'avantage par rapport à un ouvrier et coûte plus cher à l'entrepôt. De plus, même dans un cas général, il y a en tout et pour tout 3 voire 4 méthodes qui sont propres à chacun des 2 chefs (stockerL, retirerL, construireUnMeuble). Nos débats nous ont donc permis de supprimer ces classes filles et de n'utiliser dans nos stratégies que des chefs brico.

Nous avons ensuite partagé les tâches en 2 parties : la première concernait les classes correspondant à la gestion de l'entrepôt (Entrepôt, Lot, Range, Bloc) et fut codé par Soufiane Boustique. Quant à la seconde, celle-ci faisait référence à la gestion du personnel (Personnel, Chef, Ouvrier) et fut codé par Raphaël Malak.

Après avoir chacun fait sa partie de son côté, nous avons mis tout en commun et nous avons « relié les ponts » des 2 parties ensemble, en codant les fonctions qui faisaient intervenir les 2 parties. Cette liaison s'est faite majoritairement par le biais des classes *interraction* et *meuble*.

Enfin, nous avons fini par débattre et coder les différentes stratégies de gestion du personnel. Quant à la stratégie de gestion de l'entrepôt, celle-ci est unique et a été pensé en début de projet, puis codé par Soufiane Boustique. Les différentes stratégies sont expliquées ci-après, voir section III pour la gestion de l'entrepôt et V pour la gestion du personnel.

## III. Explication de la stratégie du stockage des lots dans l'entrepôt :

Il était évident à priori qu'il fallait compartimenter l'entrepôt dans ce projet selon les types de lots.

S'il y a m rangées pour n types différents, le ième type se situe dans la rangée numéro x, où x vérifie l'égalité :

$$x = i\%m\_range$$

Donc s'il y a plus de rangées que de types de lots différents, les dernières rangées de l'entrepôt seront vides. Nous avons privilégié dans ce projet le réalisme à un code « à tout épreuve ».

Chaque type de lots sera stocké dans une seule et unique rangée. Comme il peut y avoir plusieurs types différents par rangée, chacune aura un espace équivalent que nous appellerons bloc.

De plus, il est crucial que l'entrepôt soit bien ordonné, c'est-à-dire bien rangé. Nous en avons donc conclu qu'il fallait que chaque type de lots soit rangé le plus à gauche possible (toujours au sein même de son bloc), afin de pouvoir optimiser l'espace au sein de l'entrepôt et ainsi pouvoir stocker des lots de plus grand volume.

Nous avons donc décidé de tout stocker le plus à gauche possible (tout en restant dans l'espace alloué à chaque type de lots) et dans le cas où nous devons retirer un lot, que ce soit pour le supprimer ou pour l'apporter afin de construire un meuble en commande, nous avons choisi de retirer le volume requis de l'entrepôt en partant de la droite, quitte à couper le dernier lot. A première vue, cette méthode ne paraît pas efficace, seulement, sans elle, retirer un lot formerait des « trous » dans notre bloc. Il faudrait donc plusieurs ouvriers pour tout remettre en ordre (tout mettre le plus à gauche ici), ce qui nous coûterait plus cher en termes de personnels utilisés. De cette façon cela nous permet aussi de pouvoir stocker des lots de plus grand volume. Sans cette méthode, nous n'aurions pas nécessairement pu accepter des lots de volume aussi important à cause des potentiels « trous » formés par le désordre de notre entrepôt (par manque de personnel ou de temps), cela impliquerait que certains meubles ne puissent pas non plus pu être construits.

Vis-à-vis de la compartimentation, cela nous permet avec certitude d'allouer un espace minimal à chaque type de lots. De ce fait, nous sommes capables de construire des meubles qui exigent plusieurs types de lots différents voire même plusieurs meubles en même temps (dans la limite des stocks disponibles).

C'est pourquoi, en prenant compte de tous ces arguments fondamentaux, cette stratégie semble la plus efficace, elle limite l'utilisation du personnel et permet même de réduire la complexité du programme.

**Exemples :** dans le tableau ci-dessous qui représente un bloc d'un certain type de lots, une case représente un lot et le chiffre contenu son volume. Nous allons dans cet exemple simuler le nombre de personnel dont nous avons besoin pour retirer un lot de volume 7 ainsi que ceux nécessaire afin d'ordonner le bloc.

- Simulation 1 (méthode où on retire les lots suivant leur volume) :

| | | | | |
|---|---|---|---|---|
| 7 | 5 | 2 | | |
| | 5 | 2 | | 1 personnel pour retirer le lot |
| 5 | 2 | | | 2 personnels pour ordonner le bloc |

Total : 3 personnels requis

- Simulation 2 (notre méthode) :

| | | | | |
|---|---|---|---|---|
| 7 | 5 | 2 | | |
| 7 | | | | 2 personnels pour retirer le lot |

| 7 | | | | 0 personnel pour ordonner le bloc |
|---|---|---|---|---|

Total : 2 personnels requis

Nous observons bien qu'avec notre méthode, nous obtenons un meilleur résultat dans tous les cas. Si maintenant le 5 devient un 3, les 2 méthodes auraient utilisé 3 ouvriers, mais en aucun cas, notre méthode sera moins efficace. Celle-ci est de plus facile à la fois à assimiler et à implémenter. Enfin, notre méthode nous permet d'utiliser tout au plus 2 ouvriers pour stocker un lot (voir section IV la méthode 'stockerL').

## IV. Explication des méthodes principales (dans chacune des parties) :

- 'stockerL' → Rajoute un lot tout en gérant le personnel. Grâce à notre stratégie de rangement (voir partie III), cette méthode nous demandera 1 voire 2 personnels. Cette fonction va automatiquement incrémenter de 1 la disponibilité du personnel utilisé lors du stockage du lot entré en consigne. Il utilisera 2 personnels dans le seul et unique cas où il est nécessaire de retirer de l'entrepôt un lot de plus petit volume pour pouvoir stocker un plus grand lot (du même type). Si plus de 2 personnels sont obligatoires pour effectuer le rangement de ce lot, nous refusons celui-ci.

- 'construireUnMeuble' → Quoiqu'il arrive, un meuble en consigne sera ajouté à la liste d'attente. Donc même dans le cas où nous n'avons pas assez de lots pour construire un meuble, nous le rajoutons à la liste d'attente, dans l'attente de recevoir un/des lots(s) nous permettant de le construire lorsque nous 'arriverons' à celui-ci. Si la liste d'attente est actuellement vide au moment de l'exécution de la consigne, nous le refusons. Pareillement, si lorsque le personnel en charge de l'apport des différents lots nécessaires à la construction des meubles en liste d'attente arrive au meuble en question, et qu'il n'y a pas assez de lots disponibles pour le construire, nous le refusons (voir section V pour d'avantage de détails sur le fonctionnement de la liste d'attente). Lorsque nous avons assez de lots mais pas assez de personnel disponible, ce meuble va être inéluctablement construit, mais il sera mis en attente, jusqu'à ce que le personnel finisse d'acheminer les lots et qu'il y ait ensuite un constructeur de disponible, cela prendra le temps nécessaire. La liste d'attente fonctionne par chronologie d'arrivage. De cette façon, un meuble dont l'acheminement des lots a débuté sera forcément construit. Il va sans dire que nous augmentons notre trésorerie à chaque fois qu'un lot est acheminé. Il y'a enfin certaines contraintes supplémentaires pour que nous acceptions de construire un meuble (voir section V pour plus de précisions).

- 'licencier' → Cette fonction ne peut s'appeler qu'une fois par pas de temps, quel que soit le membre du personnel à licencier. Pour licencier un ouvrier, Raphaël Malak a ajouté un attribut dans la classe Ouvrier qui pour chaque objet de cette classe, pointe vers le chef de l'équipe au sein de laquelle l'ouvrier se situe. Il suffit donc d'accéder à l'équipe de cet attribut et de remplacer l'occurrence de l'ouvrier (dont nous savons à coup sûr qu'il se trouve dans l'équipe) par 'null'. Pour licencier un chef, il faut avant toute chose savoir que comme il n'est pas possible de licencier 2 membres du personnel durant la même itération, il faudra obligatoirement répartir dans les autres équipes les ouvriers d'un chef que nous souhaitons licencier. Nous vérifions donc d'abord s'il y a assez de places disponibles dans les autres équipes pour intégrer les ouvriers du chef à licencier à celles-ci. S'il n'y en a pas assez, nous ne licencions pas ce chef. Sinon, chaque ouvrier sera redirigé vers la première place disponible dans les autres équipes puis on supprimera le chef de la liste des chefs. Cette méthode retourne un booléen indiquant si le licenciement a pu se faire ou pas.

- 'recruter' → Cette fonction ne peut s'appeler qu'une fois par pas de temps, quel que soit le membre du personnel à recruter. Pour recruter un ouvrier, il faut préciser en paramètre le nom, le prénom et la spécialité (la pièce de la maison) de celui-ci. Nous vérifions d'abord qu'il y a une place disponible dans l'une des équipes des chefs actuellement dans notre personnel. S'il n'y en a pas, nous ne recrutons pas l'ouvrier. Dans le cas contraire, nous l'ajoutons à la première équipe qui n'est pas pleine (c'est-à-dire qui ne contient pas 4 ouvriers). Pour recruter un chef, il faut préciser en paramètre le nom, le prénom et la spécialité (brico ou stock) de celui-

ci, puis nous l'ajouterons à la fin de la liste des chefs déjà existante. Cette méthode retourne un booléen indiquant si le recrutement a pu se faire ou pas.

## V. Explication des différentes stratégies de recrutement du personnel :

Nous avons fait le choix dans ce projet de ne jamais faire appel à un chef Stock car ceux-ci coûtent plus chers qu'un ouvrier (10 euros par itération contre 5 pour un ouvrier) mais n'octroient aucun avantage qui leur est propre. A contrario, un chef Brico peut lui construire tout meuble, alors qu'un ouvrier ne peut construire que les meubles correspondant à sa spécialité.

De plus, nous avons vite conclu que de recruter plusieurs chefs était dans l'immense majorité des cas moins rentable que de n'en avoir qu'une seule. Nous manquions de temps pour mettre en œuvre cette dernière stratégie.

Enfin, comme expliqué précédemment, tout stockage de lot, quel qu'il soit, utilisera au maximum 2 ouvriers (voir section IV la méthode 'stockerL'). Il est donc évident qu'il est inutile de commencer avec une équipe d'ouvriers complète (c'est-à-dire avec 4 ouvriers) quand 1 suffira probablement à exécuter la première consigne réalisable (stocker un lot ici car il est impossible de construire un meuble tant que nous n'avons pas reçu de lots). Cela explique pourquoi nous ne commençons jamais avec plus de 2 ouvriers en début d'exécution des consignes.

Dans chacune des 4 stratégies retenues, le personnel sera recruté avant le début de l'exécution des consignes (quel que soit le mode d'interaction) et chacun d'entre eux aura la spécialité ''sans'' qui signifie qu'ils ne s'occuperont jamais de la construction d'un meuble. Cela nous permet de nous assurer que nous aurons toujours assez d'ouvriers pour stocker un lot. A noter également que les consignes sont traitées suivant certaines priorités : les consignes « stocker un lot » et « rien » sont exécutées avant l'exécution de la liste d'attente. L'exécution d'une consigne « meuble » est expliquée section IV.

Chaque ouvrier recruté pendant l'exécution des consignes pour gérer des lots sera licencié à la fin du tour. Un ouvrier recruté pour aider à la construction des meubles sera licencié dès que possible après qu'il ait fini de le construire (il ne peut y a voir qu'un licenciement par tour).

Les 2 premières stratégies ci-après pourront être exécutées avec au départ 1 ou 2 ouvriers (au choix de l'utilisateur en début de programme).

- **Stratégie 1 :** Complexité : ★

Il s'agit ici de la stratégie de gestion du personnel la plus basique qui soit. Le personnel sera composé d'un Chef dont l'équipe sera d'un ou de 2 ouvriers (leurs attributs seront générés par le programme) au choix de l'utilisateur. Cette équipe sera fixe tout au long de l'exécution des consignes et sera vidé à la fin de celle-ci.

- **Stratégie 2 :** Complexité : ★★★

Comparé à la dernière stratégie, nous avons implémenté à celle-ci deux améliorations. Tout d'abord, nous avons ajouté une condition de rentabilité à la construction d'un meuble, celle-ci suit la formule :

10*dureeConstruction + 5*nb <= Prix de revient du meuble

Où nb fait référence au nombre d'ouvriers requis pour apporter tous les lots nécessaires à la construction du meuble.

De plus, dans le cas où il faudrait 2 ouvriers pour stocker un lot ou bien le chef d'équipe est disponible mais que la liste d'attente n'est pas vide (voir section IV méthode construireUnMeuble pour l'explication de la liste d'attente), nous recrutons un ouvrier en plus qui soutiendra le ou les 2 autre(s) à stocker le lot ou à apporter les lots pour la construction des meubles. Cet ouvrier sera de spécialité ''sans''. A rappeler qu'il n'est possible de recruter qu'un seul personnel par pas de temps.

Cette stratégie est toujours plus rentable que la première.

➢ **Stratégie 3 :** Complexité : ★★★★★

Cette stratégie possède toutes les spécificités de la deuxième, mais par rapport à cette dernière, si aucun recrutement n'a été fait après l'exécution de la consigne à un certain pas de temps, et qu'il y a un meuble qui peut être construit (tous ses lots ont été apportés) en moins de temps que la disponibilité actuelle du chef, alors nous recrutons un ouvrier pour construire ce meuble (sa spécialité sera évidemment celle du meuble).

Contrairement aux autres recrutements d'ouvriers, nous pouvons posséder plusieurs ouvriers qui aident à la construction des meubles (2 au maximum pour ne pas empêcher le stockage d'un lot par manque de personnel).

Cette stratégie est notre stratégie la plus rentable (parfois de très loin). Dans le pire des cas, celle-ci sera aussi rentable que la stratégie 2 avec un ouvrier.

➢ **Stratégie 4 :** Complexité : ★★★★★

Cette stratégie est exactement la même que la troisième, mis à part que la méthode de rentabilité n'est pas la même. En effet, comme il est possible que ce soit un ouvrier qui construise le meuble, nous avons décidé de remplacer dans la formule de rentabilité 10*duree par 7,5*duree (c'est la moyenne de 10 et 5 : les salaires d'un chef et d'un ouvrier).

Elle est cependant moins rentable que la troisième stratégie.

## VI. Tableau de complexité des principales fonctions du projet :

- n correspond à la taille de la liste des chefs au sein du personnel. Comme vu à la section V, celle-ci est de 1, mais nous allons garder la notation n dans un cadre général ;
- m correspond au nombre de rangées ;
- mb le nombre de blocs dans la rangée du type que nous allons stocker ;
- mt la taille du bloc de ce type ;
- Ht la taille de la table de hachage (HashMap en anglais) d'un meuble (contenant la liste des lots requis) ;
- nbV le volume des lots nécessaire pour satisfaire le ième composant d'un meuble ;

**NB :** on inclut bien évidemment dans le calcul la complexité des sous fonctions utilisées dans les fonctions principales.

| Méthode | stockerL | construireUnMeuble | licencier | recruter |
|---|---|---|---|---|
| Complexité | O(m+mb+mt) | O(Ht*nbV) | O(n) pour un chef<br>O(1) pour un ouvrier | O(1) pour un chef<br>$O(n^2)$ pour un ouvrier |

## VII. Les difficultés rencontrées :

Selon nous, ce projet fut le plus dur de l'ensemble de notre cursus à Dauphine. Contrairement aux années précédentes, le sujet-ci ne nous explicitait pas toute l'architecture à suivre. La nouveauté était que les élèves cette année avaient une certaine liberté quant à la façon de coder et à la façon d'arriver à un résultat parfois équivalent.

Il y'avait de plus de nombreux paramètres à prendre en compte simultanément dans ce devoir : comment stocker dans son entrepôt, comment le gérer, quand est-ce que nous recrutons un membre du personnel et quand est-ce qu'on le licencie, comment optimiser l'utilisation du personnel, etc… La principale difficulté était que tous ces axes de travail étaient interconnectés, nous rendant difficile la conception d'une seule partie du projet sans empiéter sur d'autres.

C'est dans cette optique qu'une longue réflexion se montra fort efficace, permettant justement de construire ce réseau de classes et de méthodes grâce à un plan précis et prédéfini.

D'autre part, il fut complexe de rester motivé et confiant vis-à-vis de nos choix de stratégies, parce que quelles qu'elles soient, un contre-exemple se trouvait et mettait en lumière une autre stratégie, alors plus efficace que la dernière dans cette situation.

## VIII. Axes d'améliorations du code actuel :

- Il aurait été utile d'ajouter des images à notre projet, c'est-à-dire une interface visuelle montrant à chaque pas de temps l'état de l'entrepôt, etc… Elles pourraient être équivalentes au fichier que notre programme génère (à la demande de l'utilisateur) pour afficher l'état de notre entrepôt et de notre personnel à chaque itération effectuée.

- Nous aurions pareillement pu améliorer notre stratégie de recrutement en la poussant à son summum. Notre stratégie est optimale si les temps de construction ne sont pas immensément grands mais dans certains cas, il est évident qu'une nième équipe permettrait d'être plus productif et de maximiser d'avantage les gains de l'entrepôt (gain de temps d'exécution et donc gain d'argent). Cette stratégie est abordable mais nous manquions de temps pour la concrétiser.

**NB :** il faut nous excuser si la classe « interraction » s'écrit avec 2 « r » … Disons que nous n'étions pas très motivés pour effectuer tous les changements.

**PS :** ce ne sont sûrement pas les seules erreurs d'orthographe dans notre code…

**NB Bonus :** La réponse à la question bonus (sur la provenance de BIILIMO) n'est-elle pas Belimo (leader mondial de la technologie des servomoteurs et des vannes de régulation pour le chauffage, la ventilation et la climatisation) ?