CRAZY CHARLY DAY

PARTIE OPTIMISATION

Pour cette édition du Crazy Charly Day, le but était de développer un site web pour l'association AARS - La Boite à Cuisine, cette association proposant divers ateliers de cuisine sur différents thèmes (japonais, oriental, français, amérique du sud, etc…).

Il est donc ainsi possible pour les usagers d'émettre leurs vœux, à la manière d'un Parcoursup, c'est-à-dire de lister leurs choix par ordre de préférence, ainsi que d'indiquer le nombre d'ateliers pour lesquels ils souhaitent participer tout au plus.

Cela pose donc une question d'optimisation: comment affecter les usagers aux différents ateliers, sachant que les places sont limitées?

Pour cela, on décide d'instaurer quelques règles, pouvant aider à la résolution de ce problème:

- Suivant l'ordre de préférence, un choix rapporte plus de points qu'un autre
  En imaginant qu'une personne possède 6 choix. Son 1er choix, si accepté, rapporterait 10 points. Son 2ème choix, si accepté, rapporterait 8 points, etc…
- Si une personne est déjà affectée à un atelier, alors pour tout autre atelier auquel il serait affecté, le score obtenu sera celui du rang d'en dessous.
  Par exemple, si son 2ème choix est accepté après son 1er, alors le score obtenu sera de 6, soit le score du choix 3, ce qui fait un total de 16 points.
- Si une personne n'a aucun atelier affecté, alors un score de -10 est ajouté.
  Cela permet ainsi de favoriser le fait d'assigner à chaque personne au moins un atelier, puisque le but est d'obtenir le nombre maximum de points possibles
- Si une personne a au moins deux ateliers de moins qu'une autre, un score de -10 est ajouté
  Ceci permet d'éviter que certaines personnes aient plus d'ateliers que d'autres, bien évidemment si la personne ayant le moins d'ateliers entre les deux n'a pas atteint le nombre d'ateliers souhaités maximum.
- Une même personne ne peut pas participer au même créneau d'un atelier plus d'une fois.

A partir de ces règles, on cherche donc à obtenir le score global le plus haut possible, c'est-à-dire le maximum.

Pour cela, on peut imaginer différentes approches:

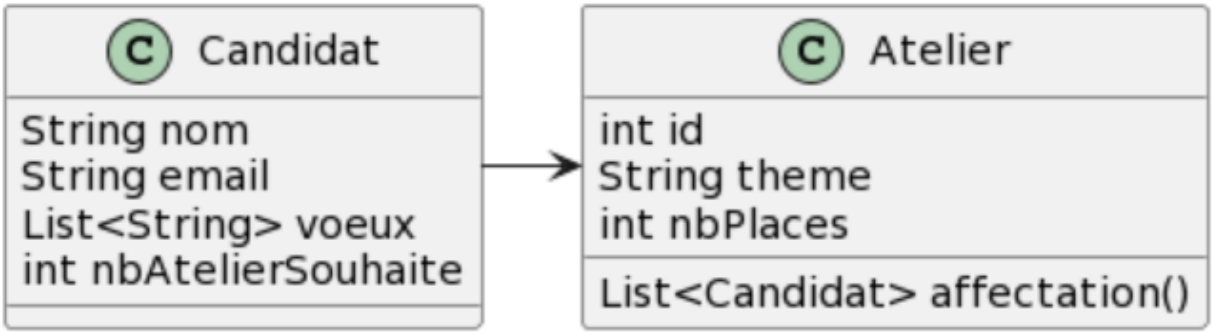
1. **Tester toutes les possibilitées**
   Cette approche permettrait de trouver la meilleure solution pour sûr, cependant elle cause plusieurs problèmes, notamment lorsque l'on parle de plusieurs ateliers avec

plusieurs places et plusieurs candidats avec plusieurs souhaits d'ateliers. En bref, plus il y a de données, plus la recherche de la meilleure solution sera longue, et de manière plutôt exponentielle. Cette solution, bien qu'elle nous permettrait d'obtenir la solution, est donc loin d'être la meilleure possible. On testerait ainsi toutes les combinaisons, et dès qu'on obtient un score plus grand que le plus haut score précédent, on remplace ce dernier par celui trouvé.

2. **Prioriser par rapport aux préférences des candidats**
   On pourrait aussi essayer de prioriser les affectations par rapport aux choix et préférences des candidats. Par exemple, on regarde le nombre de places disponibles pour un atelier, puis on regarde les souhaits des candidats. Le candidat ayant le choix de cet atelier à la plus haute préférence parmi les autres candidats sera donc priorisé pour cet atelier, et ainsi de suite.
   Les problèmes venant de cette solution serait qu'on ne choisit pas en fonction du score, et donc qu'il n'y a pas de garantie que cela soit la meilleure solution (on pourrait se retrouver avec des cas où pour l'un des candidats, son 6eme choix a été choisi, ce qui n'est vraisemblablement optimal. Pour d'autres cas, on aurait des candidats sans ateliers, et aussi des candidats ayant plus d'ateliers que d'autres, ce qui engendre des points en moins.
3. **Choisir une configuration aléatoire, et alterner certains choix**
   Cette solution se base sur le fait d'assigner aléatoirement les ateliers aux candidats, puis de regarder le score, et ensuite échanger deux ateliers entre deux candidats, et à nouveau regarder le score. Si le score obtenu est supérieur, alors cette configuration est gardée. Sinon, on continue jusqu'à obtenir un score meilleur. Si l'on ne trouve plus d'amélioration, alors on aurait vraisemblablement trouvé le meilleur score et ainsi la meilleure combinaison possible.
   Cette méthode peut être efficace, mais elle se repose beaucoup sur l'aléatoire, et peut ainsi prendre du temps. Néanmoins, il y a de fortes chances que cette solution reste le plus souvent plus rapide que la première, qui était de tester toutes les possibilités.

De ces trois premières solutions, la plus intéressante serait celle dite "aléatoire". Cependant, il existe très certainement des algorithmes de recherches plus performants et rapides, qui n'ont malheureusement pas été trouvés.

Pour ce qui est de la représentation de ces données, en Java, on pouvait imaginer ces classes:



Cette partie a été assez difficile, du fait que notre groupe n'était composé que de 4 étudiants, dont une seule RA-IL. Il existe très certainement des algorithmes plus optimisés pour ce problème, mais nous n'avons pas réussi à les repérer.

Le manque d'étudiants dans ce groupe a aussi posé problème par rapport à la programmation de ces algorithmes, qui n'a malheureusement pu être réalisée.