M1 Informatique - Ingénierie Logicielle
TD-TP
Séparer l'émission de commandes de leur exécution.

Prétexte

Un exercice classique donné en mathématiques au collège consiste à chercher à obtenir un volume d'eau donné, à partir d'un certain nombre de bidons dont un connait le volume, et d'une source d'eau (supposée inépuisable). Lors du jeu, on peut remplir un bidon (à ras bord), le vider (complètement), ou le transvaser dans un autre (on met dans le deuxième bidon soit la totalité du volume du premier bidon si c'est possible, soit sinon le volume exact nécessaire pour remplir complêtement le deuxième bidon). On suppose toujours être capable d'effectuer ces opérations sans pertes liées à des éclaboussures, ou au fait qu'il reste toujours des gouttes d'eau au fond d'un récipient que l'on vient de vider.

Informatisation

On souhaite réaliser un simulateur de ce jeu, permettant de créer un jeu à n bidons de contenances diverses, dans lequel un joueur peut exécuter les actions précédemment listées afin d'obtenir un volume donné d dans le premier bidon de la partie de jeu en cours (Il sera convenu que le premier bidon d'une partie à une contenance supérieure ou égale à d).

```
int[] capacitesBidons = {200, 100, 50};
Partie1 p =
    new Partie1(
        3, //nombre de bidons
        capacitesBidons, //capacité des bidons
        150); //volume à atteindre
p.jouer();
```

La méthode `jouer()` simule le comportement interactif d'un joueur et les actions qu'il exécute, ou plus exactement les commandes d'actions qu'il émet. Chaque commande d'action entraîne l'exécution d'une méthode correspondant à l'action. Quand le volume recherché est atteint, la partie est gagnée et il est alors possible d'afficher la suite d'actions qui a été réalisée pour atteindre le but; la suite d'actions n'a pas à être optimale pour que la partie soit gagnée. On pourra par exemple avoir :

```
  Gagné avec la solution : [RemplirBidon-2, TransvaserBidon-2-1,
  RemplirBidon-2, RemplirBidon-3, TransvaserBidon-3-1]
```

Contraintes

On souhaite pouvoir revenir en arrière (undo) en défaisant la dernière action effectuée, et ce éventuellement plusieurs fois de suite. Un *undo* défait la dernière action réalisée a2 et un nouvel *undo* consécutif défait l'action a1 réalisée avant a2, même si a1 était elle-même un *undo*.

On peut souhaite connaître à l'issue d'une partie la séquence d'action qui a été réalisée pour arriver au but incluant, ou pas, les éventuels *undo*.

On souhaite pouvoir rejouer dans une nouvelle partie, une séquence d'actions enregistrée lors d'une partie précédente.

On peut imaginer que l'ordinateur est connecté à un robot qui réalisent les actions. Le robot n'est pas toujours disponible, on peut souhaiter enregistrer une suite d'actions pour la faire exécuter plus tard par le robot.

Il est à noter que ces contraintes sont en fait extrêmement classiques. Presque tous les logiciels interactifs satisfont certaines d'entre elles (penser au générique ctrl-z "undo").

Proposez une solution pour ce problème. Vous pourrez donc vous demander si une architecture logicielle type solutionnant ce problème récurrent a été proposée, et si oui l'étudier.

Améliorations

- Comment faire de Undo une commande? On pourrait ainsi avoir une trace qui serait :

```
Gagné avec la solution : [RemplirBidon-2, TransvaserBidon-2-1,
RemplirBidon-2, TransvaserBidon-2-1, undo-last, RemplirBidon-3, TransvaserBidon-3-1]
```