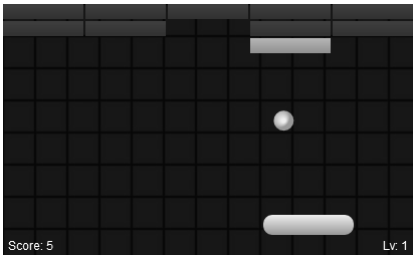
UE Conception Orientée Objet

Casse Briques

On s'intéressera dans cet exercice à la modélisation d'un certain nombre d'entités d'un jeu de casse-briques (*breakout-style game*).
Ce jeu se joue seul et le joueur contrôle une raquette (*paddle*) qu'il peut déplacer latéralement. La raquette permet de renvoyer une balle (*ball*) vers des briques (*blocks*) qui sont cassées lorsqu'elles sont frappées par la balle, tout en renvoyant la balle. Le but est de détruire toutes les briques sans laisser tomber la balle. Le joueur dispose de plusieurs balles en réserve qu'il peut utiliser lorsqu'il en perd une lui donnant ainsi quelques droits à l'erreur.



Ce jeu d'arcade fait partie des ancêtres des jeux vidéos et a connu des variantes dans lesquelles les briques sont de différents types et déclenchent des effets variables lorsqu'elles sont détruites.
La classe `Player` modélise l'entité joueur du casse-briques, avec notamment sa raquette (la taille pouvant varier) et ses balles en réserve.
Les briques sont caractérisées par un nombre de points (reçus par le joueur lorsqu'il casse la brique) et par l'effet que leur destruction a sur le joueur.
On trouve les types de briques suivants :

- les briques *simple* rapportent 100 points et n'ont aucun effet particulier autre qu'ajouter les points au score du joueur ;
- les briques *jackpot* rapportent aléatoirement de 1 à 10 fois plus de points qu'une brique de base et n'ont aucun effet particulier ;
- les briques *grande raquette* rapportent autant qu'une brique simple et doublent la taille de la raquette ;
- les briques *bonus balle* rapportent 3 fois plus qu'une brique simple et augmentent de 1 le nombre de balles en réserve du joueur ;
- les briques *piège* rapportent 0 point et ont pour effet d'enlever un nombre de points variable, précisé à la construction de la brique, et de supprimer une balle de la réserve du joueur (s'il lui en reste).

D'autres types de briques doivent pouvoir être envisagés et ajoutés facilement.

Q 1 . Dans cette version du jeu il n'y a nécessairement qu'un seul joueur. Une seule instance de la classe `Player` est donc nécessaire. Comme prendre en compte cela dans la conception de cette classe ?

Donnez un diagramme UML pour la classe `Player`.

Q 2 . Donnez un diagramme de classes UML pour représenter les briques.

Q 3 . On propose d'enrichir les briques possibles : pour chaque type de brique déjà proposé, une variante est créée. Cette variante appelée *double effet* (*double-effect block*) a comme conséquence de doubler le nombre de points attribués pour chacun des types ainsi que l'effet appliqué. Ainsi :

- une brique *bonus balle double effet* rapportera 6 fois plus qu'une brique simple et 2 nouvelles balles pour le joueur ;
- une brique *jackpot double effet* rapportera aléatoirement de 2 à 20 fois plus de points qu'une brique de base et n'aura pas d'autre effet.

Évidemment pour chaque nouveau type de brique que l'on peut imaginer on veut pouvoir en avoir une version double effet.

Adaptez votre proposition de conception pour prendre en compte les briques *double effet*.

Que se passe-t-il si on ajoute ultérieurement un nouveau type de briques ?

Q 4 . Donnez le code Java pour les briques *double effet*.

Q 5 . Donnez le code JAVA permettant de créer une *brique bonus balle double effet*.

Q 6 . À chaque type de brique correspond une représentation graphique différentes (couleur, forme, ...) disposant d'une méthode l'affichant sur un canevas. Ainsi à partir d'une liste de briques (décrivant un niveau par exemple), il doit être possible de créer la liste des représentations graphiques de ces briques avec, par exemple, un code comme celui-ci :

```
List<Block> l = ...; // creating and then filling list l with blocks
List<BlockView> lg = new ArrayList<>();
for (Block block : l) {
  lg.add( ... );     // you have to write something instead of ...
}
```

Reprenez et mettez à jour votre diagramme UML pour prendre en compte cette contrainte.

Q 7 . Donnez les lignes de code permettant de créer la liste des représentations graphiques demandées.

Q 8 . Pour pouvoir garder un modèle (les briques, le joueur, la raquette) stable lorsque la représentation graphique change, il est important de **ne pas introduire de dépendance de la partie modèle vers la partie graphique**.

La solution que vous avez proposée introduit-elle une dépendance entre les modèles de briques et leurs vues ?

Q 9 . Si oui, proposez une solution corrigeant ce problème.

Vous accompagnerez votre solution d'une nouvelle version des lignes de code créant la liste des vues.

A l'issue de votre solution, il doit donc être possible de modifier l'aspect graphique d'un type de briques sans toucher au code la classe de brique associée.