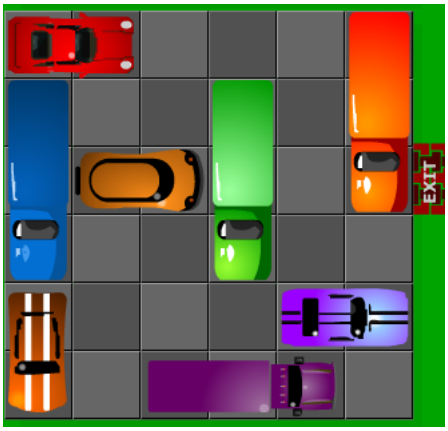
UPMC/Licence/Info/2I013
Rushdroid
Modèle du jeu

Janvier 2016

Inspiré du jeu de taquin, *blocked in* ou *rush hour* est un casse-tête de déplacement.

On dispose des pièces rectangulaires sur une grille. Le but du jeu est de déplacer une pièce distinguée vers la «sortie». Les déplacements suivent l'axe vertical ou horizontal, respectivement, vers le bas ou vers le haut ; vers la gauche ou vers la droite. La pièce distinguée ne peut sortir que par un déplacement horizontal vers la droite. Pour libérer le chemin vers la sortie il est nécessaire de déplacer également d'autres pièces.



source : `http://www.jeu.fr/jeu/rush_hour_2`

Les pièces sont de taille 2 ou 3 selon qu'elles occupent 2 ou 3 cases de la grille. Elles sont orientées soit verticalement soit horizontalement. Elles se déplacent selon l'axe de leur orientation. Le nombre de pièce n'est pas fixé, mais il y a toujours au moins la «pièce distinguée».

Le modèle du jeu permet de représenter *l'état du jeu* et d'en extraire de l'information. par exemple : la partie est-elle finie ? Il permet également de modifier l'état du jeu en réalisant des *actions de jeu*.

Les données qui représentent l'état du jeu reposent sur :

1. une modélisation des pièces du jeu ;
2. une modélisation de la grille du jeu.

Le modèle est implémenté par un ensemble de classes JAVA.

1 Classes utilitaires

Les classes du modèle du jeu devront faire partie du *package* nommé `android.rushdroid.model`

Orientation des pièces

L'orientation des pièces est représentée par le type énuméré `Direction` qui définit les deux valeurs : `HORIZONTAL` et `VERTICAL`.

Position

La position d'une pièce, et, plus généralement, les coordonnées d'une case sont représentées par une paire d'entiers : numéro de colonne, numéro de ligne. Une position peut être modifiée par changement de son numéro de ligne ou de colonne. Les positions sont réalisées par la classe `Position` dont l'interface est la suivante :

`Position(int ncol, int nlig)` constructeur, avec `ncol` numéro de colonne et `nlig`, numéro de ligne.

`int getCol()` donne le numéro de colonne.

`int getLig()` donne le numéro de ligne.

`Position addCol(int d)` donne la position obtenue en ajoutant `d` au numéro de colonne.

`Position addLig(int d)` donne la position obtenue en ajoutant `d` au numéro de ligne.

2 Les pièces du jeu

On attribue à chaque pièce un numéro. Chaque numéro est propre à une pièce. Dans une partie, deux pièces distinctes ne peuvent pas avoir le même numéro.

Par convention, la pièce qu'il faut ≪sortir≫ portera toujours le numéro 0 (zéro).

Les pièces possèdent également les caractétistiques suivantes :

- taille de la pièce : 2 ou 3 ;
- orientation de la pièce : horizontale ou verticale ;
- position de la pièce sur la grille : les coordonnées de la case *minimale* occupée par la pièce. C'est-à-dire : la case la plus à gauche pour une pièce horizontale ; la case la plus haute pour une pièce verticale.

Les pièces sont modélisées par la classe `Piece`.

Interface de la classe `Piece`

`Piece(int id, int size, Direction dir, int ncol, int nlig)` contructeur, avec

`id` identificateur

`size` taille

`dir` orientation
`ncol`, `nlig` position

`int getId()` donne l'identificateur de la pièce.

`Direction getDir()` donne l'orientation de la pièce.

`int getSize()` donne la taille de la pièce. Les valeurs obtenues doivent toujours être 2 ou 3.

`Position getPos()` donne la position de la pièce.

`void setPos(Position pos)` modifie la position de la pièce. La nouvelle position est `pos`.

3 La grille du jeu

La grille du jeu est une table de 6 colonnes et 6 lignes. Les cases de la table sont identifiées par leurs coordonnées : numéro de colonne, numéro de ligne. La première case de la première ligne a les coordonnées $(0, 0)$; la dernière case de la dernière ligne a les coordonnées $(5, 5)$. La ≪sortie≫ de la grille est la case de cordonnées $(5, 2)$.

Les cases de la grille contiennent, soit des identificateurs de pièce (type `int`), soit *rien*, lorsqu'aucune pièce n'occupe la case.

La grille de jeu est réalisée par la classe `Grid`.

Interface de la classe `Grid`

`Grid()` constructeur. Crée une nouvelle grille, de taille 6 par 6 dont aucune case n'est occupée.

`boolean isEmpty(Position Pos)` donne la valeur `true` si la case en position `pos` n'est pas occupée, `false`, sinon.

`int get(Position pos)` donne l'identificateur de la pièce qui occupe la case en position `pos`, si elle existe.

`void set(Position pos, int id)` affecte l'identificateur `id` à la case en position `pos`.

`void unset(Position pos)` libère la case en position `pos`.

4 Le modèle de jeu

L'état du jeu est défini par l'ensemble des pièces d'une partie avec leurs positions respectives (classe `Piece`) et une grille de jeu avec l'information d'occupation de ses cases (classe `Grid`).

L'état du jeu évolue en cours de partie selon les déplacements que le joueur imprime aux pièces. Selon son orientation, une pièce peut être déplacée soit horizontalement, soit verticalement. De plus, le déplacement peut être effectué, soit *en avant*, soit *en arrière*. Les quatre possibilités de déplacement sont :

— une pièce horizontale qui se déplace en avant, se déplace vers la droite ;
— une pièce horizontale qui se déplace en arrière, se déplace vers la gauche ;
— une pièce verticale qui se déplace en avant, se déplace vers le bas ;
— une pièce verticale qui se déplace en arrière, se déplace vers le haut.

On peut donc dire qu'il y a deux actions de jeux :
— faire avancer une pièce ;
— faire reculer une pièce.

Lorsque la pièce distinguée occupe la position de sortie, avec a bonne orientation, on dit que l'on est en *fin de partie*.

Le modèle du jeu doit permettre de
— créer une nouvelle partie par initialisation des pièces et de la grille ;
— réaliser les actions de jeu, avec pour effet de modifier la position de la pièce déplacée et l'occupation des cases de la grille ;
— déterminer si la partie est finie ou non.

Le modèle du jeu est réalisé par la classe `Model`.

Interface de la classe `Model`

`Model()` constructeur. Construit une partie avec pour état initial celui représenté par l'image donnée en introduction.

`Model(String fname)` constructeur. Construit une partie avec pour état initial celui donné dans le fichier `fname`. (Sera précisé ultérieurement).

`int getIdByPos(Position pos)` accesseur. Donne l'identifiant de la pièce à la position `pos` ou ≪rien≫ si la case est inoccupée.

`Direction getDir(int id)` accesseur. Donne l'orientation de la pièce dont l'identificateur est `id`.

`int getLig(int id)` accesseur. Donne la ligne sur laquelle se trouve la pièce dont l'identificateur est `id`. Si la pièce est verticale, la ligne est celle de la case la plus haute.

`int getCol(int id)` accesseur. Donne la colonne sur laquelle se trouve la pièce dont l'identificateur est `id`. Si la pièce est horizontale, la colonne est celle de la case la plus à gauche.

`bool endOfGame()` donne `true` si l'on est en fin de partie, `false`, sinon.

`void moveForward(int id)` déplacement en avant de la pièce dont l'identifiant est `id`. Affecte la position de la pièce déplacée et l'état de la grille de jeu si le déplacement est possible ; ne modifie rien sinon.

`void moveBackward(int id)` déplacement en arrière de la pièce dont l'identifiant est `id`. Affecte la position de la pièce déplacée et l'état de la grille de jeu ; ne modifie rien sinon.