HEPIA

[JavaCrush]

Java / POO

Mehmed Blazevic
21/12/2016

Contenu

Introduction

Le but de ce projet est de créer un jeu de type CandyCrush. Le principe du jeu est simple : il y a une grille avec des formes (des animaux dans notre cas) et le but est de les aligner en ligne ou en colonne. Lorsqu'on aligne plus de 2 animaux, horizontalement ou verticalement, ceux-ci disparaissent et de nouveaux sont générés aléatoirement. Le jeu doit avoir une architecture multi-threadée.

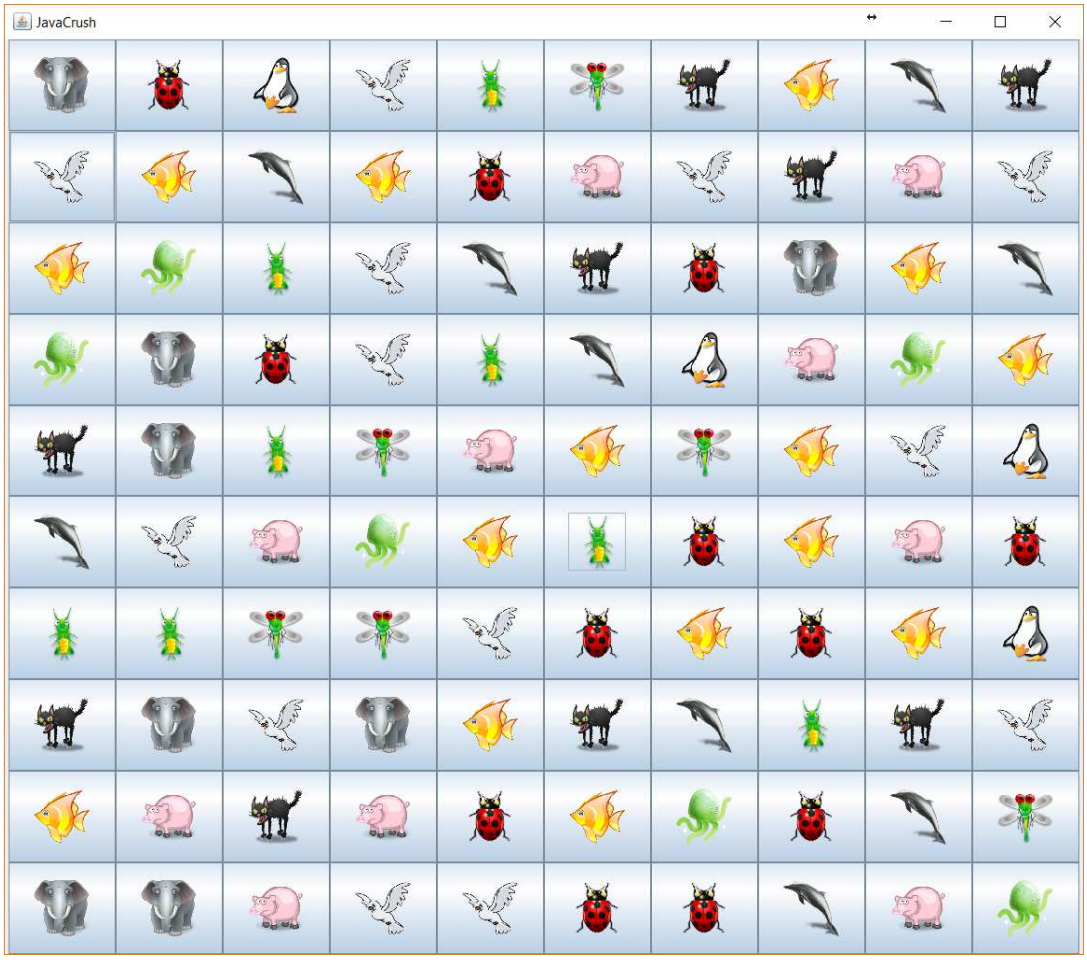
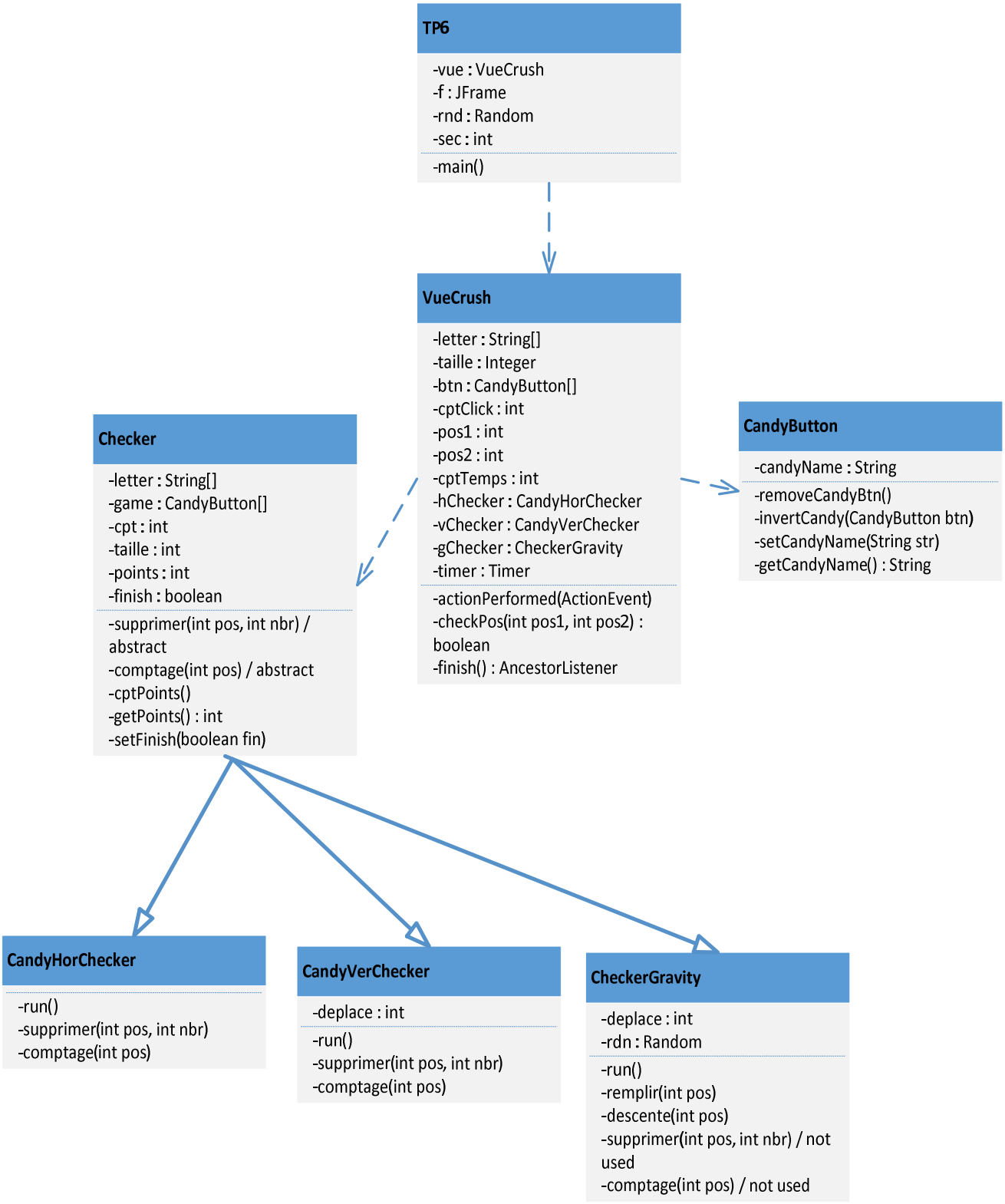


Diagramme de classes



TP6 est le point d'entrée du programme (main). Cette classe va créer la classe VueCrush. Cette dernière va créer un tableau de CandyButton et les 3 sous-classes de Checker. Le tableau représente le plateau de jeu et les 3 checkers sont les threads qui vont scruter le jeu.

Fonctionnement

Pour commencer, il y a le main() qui va créer VueCrush qui sera le coordinateur du jeu. Ce dernier, va créer à son tour un tableau de la classe CandyButton qui sera le plateau qui contient les animaux. Pour finir, le jeu se compose de 3 threads principaux : le thread qui vérifie les positions verticales, celui qui vérifie les positions horizontales et le dernier que l'on nomme le thread de « gravité ». Ces trois threads ont des rôles bien définis et c'est grâce à eux que le jeu fonctionne correctement.

VueCrush

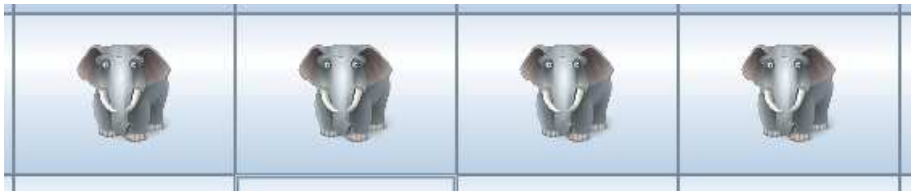
La classe VueCrush est une sous-classe de JPanel. Elle va nous permettre d'initialiser et de faire fonctionner tout le jeu. Elle va créer les threads, gérer le déplacement des cases lorsque le joueur veut bouger des animaux et va nous permettre de quitter le programme proprement en envoyant un signal de fin aux threads. De plus, un timer est implémenté, afin de mettre fin à la partie après un certain temps. Il met fin à la partie en fermant le JFrame qui contient tout le jeu. Il n'y a pas d'algorithme particulier. Il y a un « AncestorListener » qui va permettre de quitter le jeu proprement, soit en cliquant sur la croix, soit lorsque le temps imparti est écoulé.

CandyButton

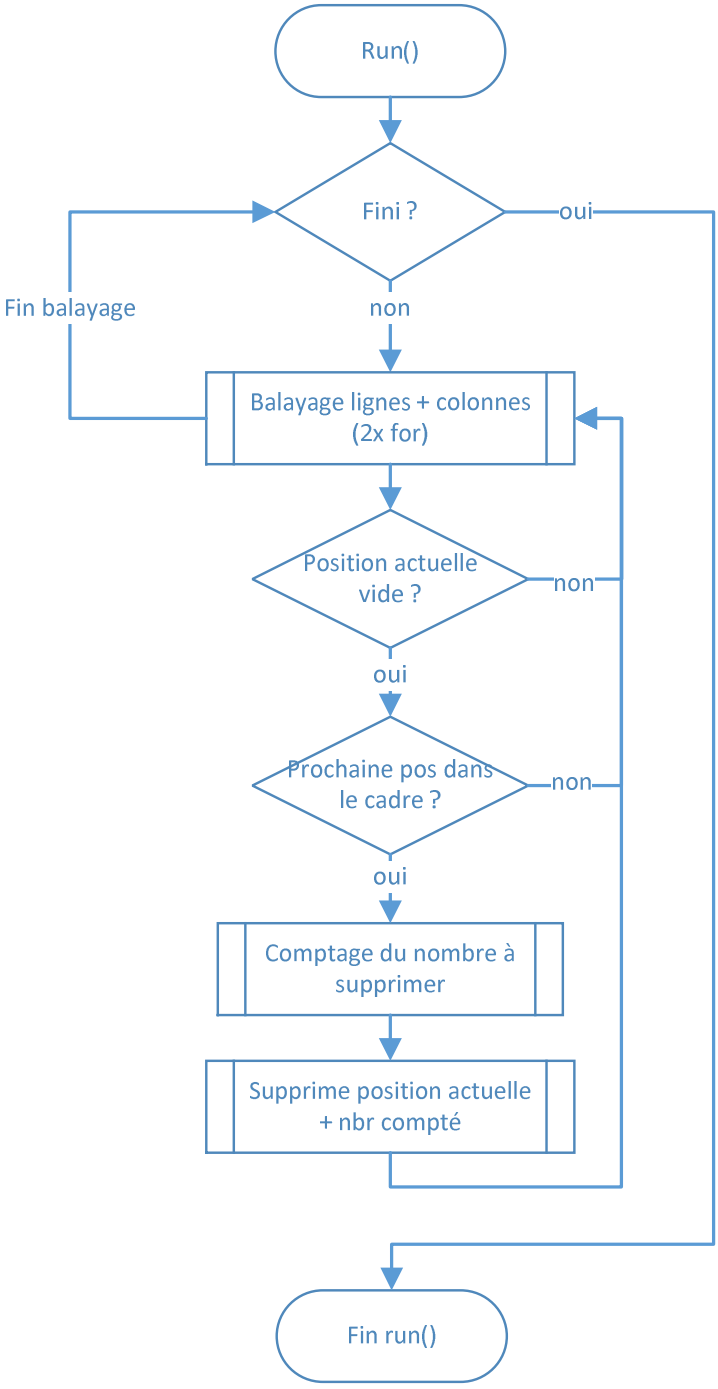
CandyButton est une classe dérivée de JButton. A la différence près qu'elle possède un champ « candyName » qui représente le nom de l'animal. C'est comme ça que la distinction entre les différentes cases est faite. Cela permet de faciliter la logique du jeu, car le nom d'un CandyButton est la position à laquelle il se trouve (de 0 à 99).

Thread vertical et horizontal

Ces deux threads fonctionnent de la même manière. Ils vont balayer tout le plateau de jeu et lorsqu'ils tombent sur les mêmes animaux, ils vont supprimer toutes les mêmes cases côte à côte.



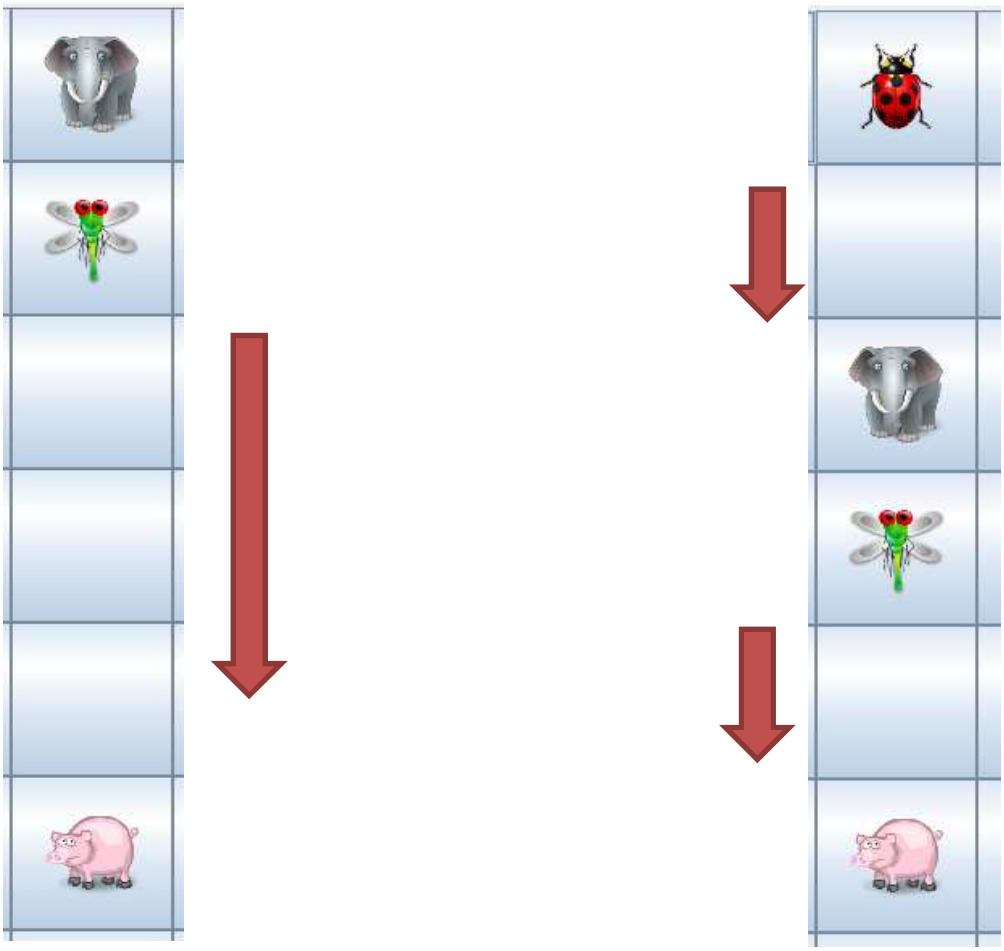
Ces deux threads ont la même logique, à la différence que l'horizontal va supprimer les animaux sur la même ligne et le vertical va supprimer les animaux sur la même colonne. Toutes les cases sont vérifiées par ces threads. À chaque suppression, un nombre de points est incrémenté. Plus il y a d'animaux supprimés en un coup, plus le joueur cumule de points.



Pour un souci de lisibilité, j'ai représenté les boucles for qui passe en revue les cases par une fonction/processus « Balayage ». C'est faux, mais ça évite d'alourdir le schéma. Le balayage se fait de haut en bas et de gauche à droite.

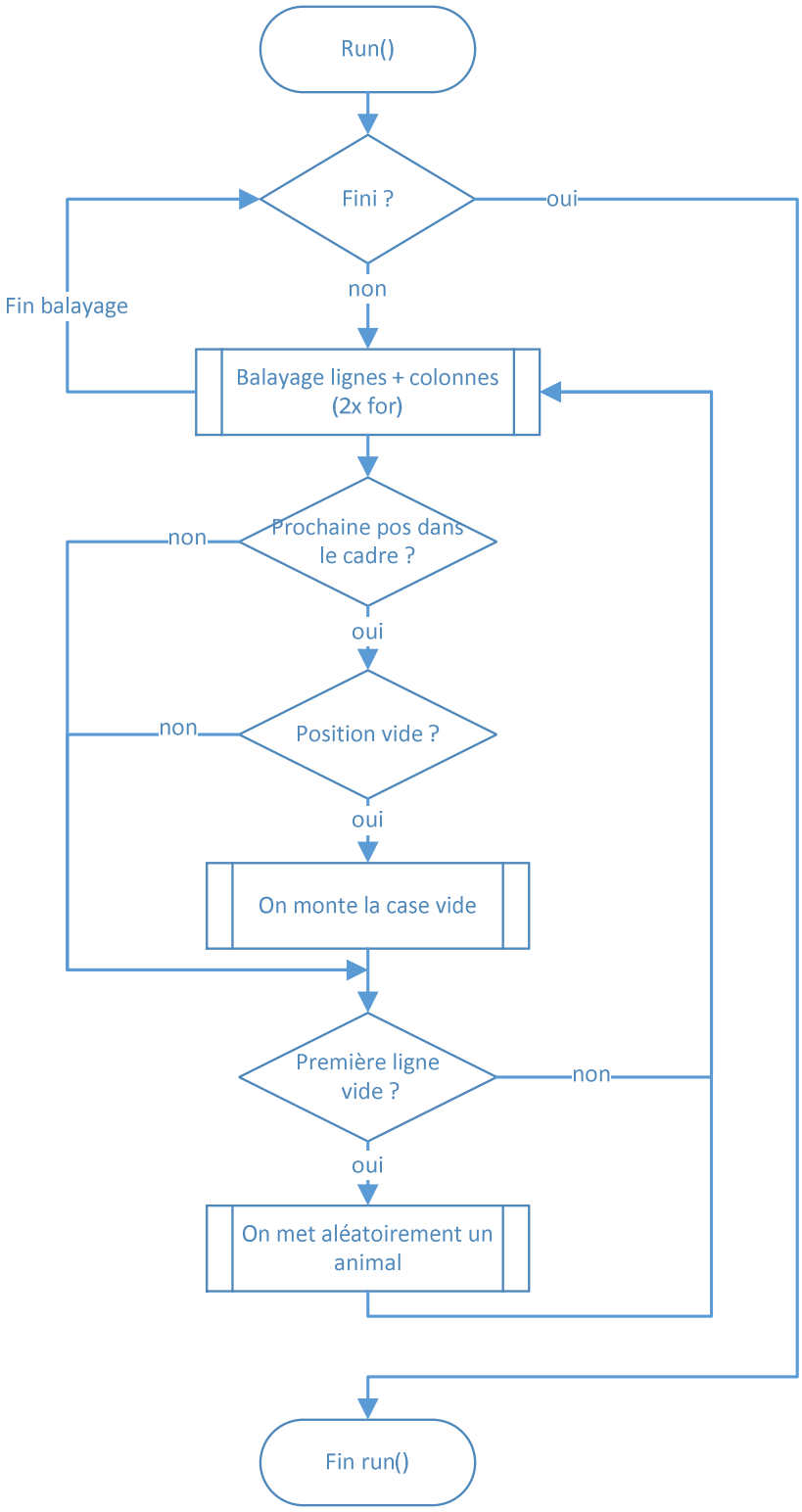
Thread gravité

Ce thread permet de descendre les images lorsqu'il y a des boutons vides. Il va aussi détecter, lorsqu'il est à la première ligne, si celle-ci est vide. Si tel est le cas, il va générer aléatoirement un animal, dans la première ligne et il va continuer à descendre tous les animaux, etc.



Une fois que la colonne est pleine, il va passer à la colonne voisine. Cela permet d'avoir une plateforme de jeu constamment pleine. Ce thread balaye le jeu de bas en haut et de gauche à droite.

Voici le fonctionnement logique de la gravité :



Les 3 threads sont en réalité des classes qui implémentent « Runnable ». Elles sont issues de la classe « Checker » qui définit un cadre général pour ces trois sous-classes. Checker définit les champs et les méthodes qui sont utilisées ou qui doivent être implémentées dans ses sous-classes, afin de garantir le bon fonctionnement du jeu.

Synchronisation

La synchronisation est le point le plus important du TP. Etant donné qu'il faut utiliser 3 threads qui ont chacun accès au tableau de boutons, il faut impérativement protéger ce dernier. Deux problématiques se présentent, la première est qu'il faut protéger le tableau en écriture. C'est quelque chose que l'on comprend instinctivement. La deuxième problématique est la protection du tableau en lecture aussi. Il est impératif de le faire, car si un thread est en train de lire les cases, pendant qu'un autre fait une suppression, des données seront croisées. On risque de supprimer une mauvaise case. C'est pour ça qu'il est impératif de protéger le tableau dans toutes les fonctions qui lisent/écrivent dedans.

Dans mon cas, je n'utilise ni wait() ni notify(). C'est un choix arbitraire, j'ai fait des tests avec et sans et j'en suis arrivé à la conclusion que je n'en avais pas besoin. Premièrement, ça rajoute du code, donc de la complexité, même si elle est relative. Deuxièmement, si je les utilise, je dois réveiller les threads séquentiellement après chaque échange de boutons, le problème est que je ne contrôle pas l'ordre de réveil des threads étant donné que je me synchronise sur le plateau de jeu « game ». Si le thread de gravité est le premier réveillé, on risque de ne pas descendre et générer de nouvelles cases, car les threads qui suppriment, passent après la gravité.
Ça peut se discuter, d'autres façons de faire peuvent être employé, j'en suis conscient. Selon moi, l'avantage est que c'est plus simple et que du coup, on est sûr que nos threads tournent en arrière-plan. Ils ne se retrouveront jamais bloqué. Le seul problème dans mon cas, est qu'il faut endormir les threads un court instant pour éviter que l'ordinateur brûle, car ils tournent constamment.

Conclusion

Ce TP est très intéressant, notamment car il s'agit d'un jeu et que l'on peut y jouer une fois fini. La grosse difficulté aura été de comprendre comment synchroniser les threads entre eux. Dû au manque de temps, le TP n'est pas aussi aboutit que je l'aurais souhaité. Je pensais ajouter des effets lors des transitions et éventuellement ajouter la possibilité de faire des combos (alignement ligne + colonne par exemple). Il n'y a pas de problèmes connus à ce jour, de nombreux tests ont été effectués. Le choix d'utiliser des threads n'est pas très judicieux pour ce TP, cela dit nous aurons vu comment les mettre en place et les synchroniser. Si le temps me le permet, je ferai un portage sur Android durant les vacances.