BUFFO Pierre

Rapport

21.12.2016

JavaCrush

# Introduction

JavaCrush est une imitation du célèbre « CandyCrush ». Jeu qui se compose d’une grille de cases et qui a pour but d’aligner des motifs identiques pour gagner des points. Le tout dans un certain laps de temps. 3 cases alignées commencent à générer des points, il est donc possible d’« éclater » jusqu’à 5 cases simultanément. La distribution des points est la suivante :

* 3 images alignées 50 points
* 4 images alignées 150 points
* 5 images alignées 400 points

# Fonctionnement général

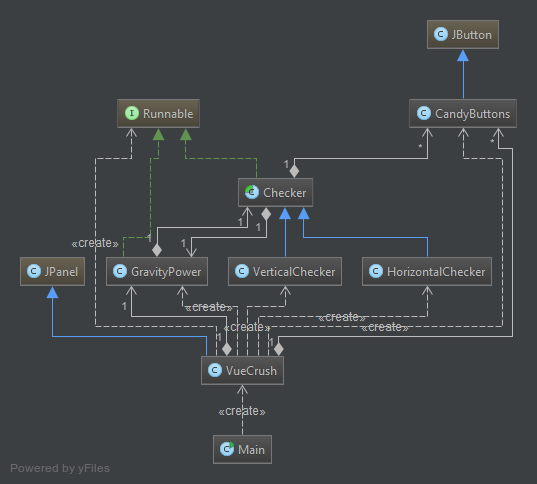
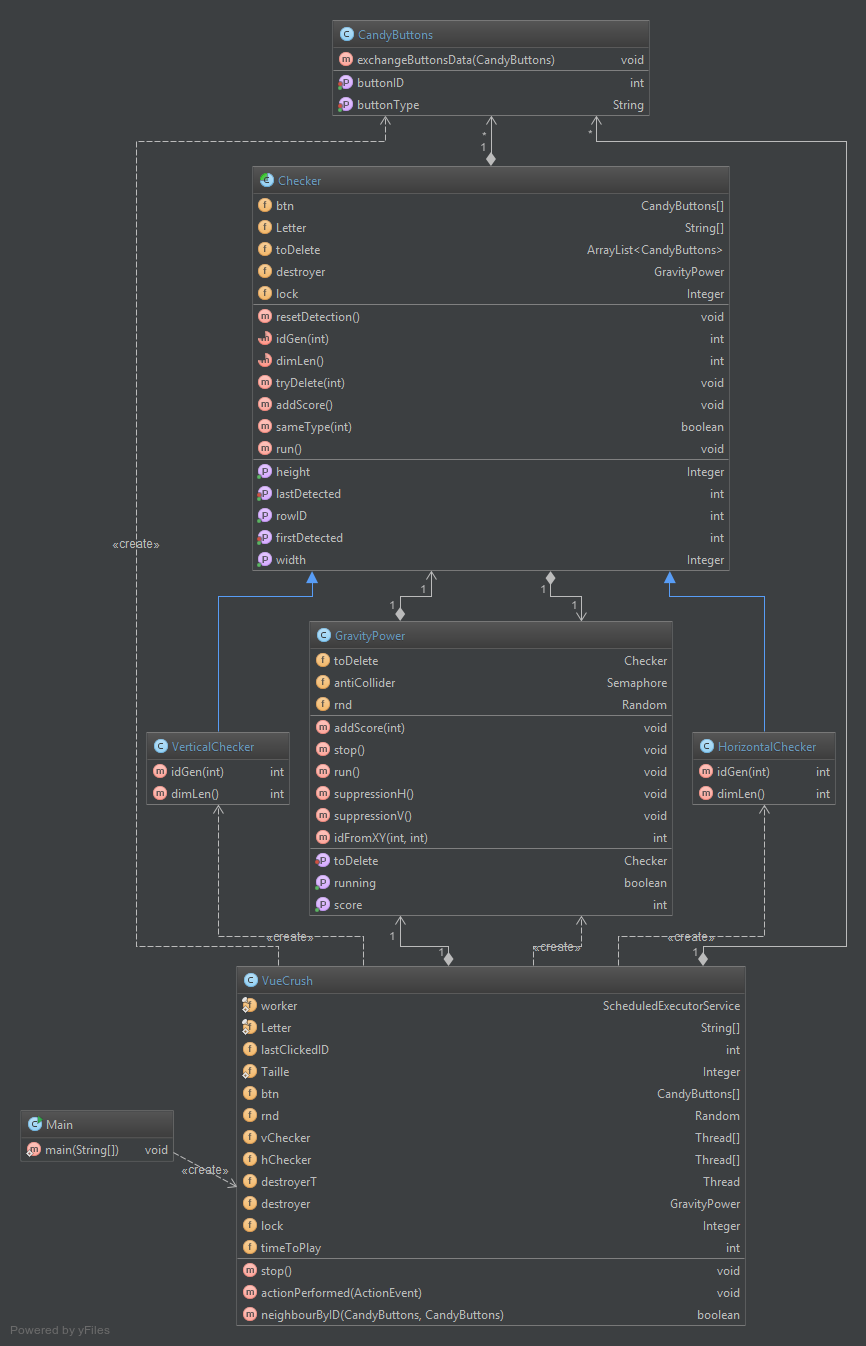
Le programme dispose d’un ensemble de boutons. Ces derniers sont échangés de positions si l’utilisateur effectue une demande de déplacement valide (distance d’une case horizontale ou verticale).

Le programme dispose d’un thread vérificateur par ligne et par colonne. Ces threads de classe HorizontalChecker ou VerticakChecker sont dérivés de la classe abstraite Checker. Ces threads, regardent en permanence l’état du plateau, dès que 3 cases identiques sont alignées (de manière horizontale ou verticale) le checker concerné va alors augmenter le score puis s’envoyer lui-même dans l’objet de destruction.

L’objet de destruction « GravityPower » est un thread lui aussi, dès qu’il reçoit un objet de type checker il va procéder à l’élimination des cases, puis appliquer un effet de gravité et faire « tomber » de nouvelles cases.

De manière générale tous les threads tournent en boucle jusqu’à ce que la variable isRunning contenue dans l’objet GravityPower passe à false.

# Diagramme de classes



# Implémentation

Description des membres et méthodes des classes

## CandyButtons

CandyButtons étends la classe JButton, en rajoutant des attributs :

* buttonID Contient l’identifiant unique du bouton.
* buttonType Contient le type du bouton (nom de l’image avec extension).

Ainsi qu’une méthode :

* exchangeButtonsData Qui échange le buttonType ainsi que l’icones de deux boutons.

Les autres méthodes sont tout ce qu’il y a de plus standard (getter/setter/constructeur).

## Checker

Un objet de type checker est un objet chargé de vérifier la présence d’une combinaison gagnante, dès qu’une suite de 3 boutons similaire est détectée une procédure de suppression est lancée.

protected CandyButtons[] btn; liste des boutons

protected String[] Letter; liste des noms d’images

private int width, height; largeur, hauteur du jeu

private int rowID; identifiant unique du vérificateur

protected ArrayList<CandyButtons> toDelete; liste de boutons (utilisé pour la détection)

protected GravityPower destroyer; objet de destruction

private int firstDetected; premier élément détecté (utilisé pour la détection)

private int lastDetected; dernier élément détecté (utilisé pour la détection)

public Integer lock; valeur utilisée uniquement pour gérer les problèmes de synchronisation

La classe abstraite checker sert de modèle pour les checker horizontaux et verticaux. Outre le fait de contenir un constructeur ainsi que des getter/setter elle contient les méthodes suivantes :

public abstract int idGen(int id); méthode permettant de déterminer l’identifiant d’un bouton en fonction d’un identifiant, le deuxième identifiant (qui compose une paire x, y) est déterminé par l’identifiant unique d’un checker. Cette méthode variant d’un checker horizontal à vertical est donc abstraite.

public abstract int dimLen(); méthode retournant la longueur de la ligne ou de la colonne de l’objet. Cette méthode est donc abstraite.

public void addScore() ajoute du score à la partie, les points à ajouter sont calculés en fonction de lastDetected-firstDetected. Les points sont alors ajoutés dans l’objet de destruction destroyer.

public boolean sameType(int id) vérifie si deux boutons sont du même type (même image).

public void tryDelete(int id) essaye de supprimer des cases à condition que au moins 3 cases similaires aient été détectées.

public void run() contenu du thread, boucle en rond tant que l’objet destroyer n’indique pas une fin de partie.

Algorithme de fonctionnement global :



## HorizontalChecker

La classe HorizontalChecker sert à vérifier la présence de plus de deux boutons identiques alignés horizontalement. Elle hérite de la classe checker dans laquelle il est possible de préciser l’ID de la ligne à vérifier. Les méthodes abstraites suivantes sont redéfinies :

public abstract int idGen(int id); retourne la multiplication de l’identifiant de ligne \* largeur de ligne + id en paramètre, afin d’obtenir l’identifiant d’une case.

public abstract int dimLen(); retourne la largeur du plateau de jeu.

## VerticalChecker

La classe VerticalChecker sert à vérifier la présence de plus de deux boutons identiques alignés verticalement. Elle hérite de la classe checker dans laquelle il est possible de préciser l’ID de la colonne à vérifier. Les méthodes abstraites suivantes sont redéfinies :

public abstract int idGen(int id); retourne la multiplication de l’id\* largeur de ligne + l’identifiant de colonne en paramètre, afin d’obtenir l’identifiant d’une case.

public abstract int dimLen(); retourne la hauteur du plateau de jeu.

## GravityPower

La classe GravityPower est en charge de détruire les cases détectées par les threads checkers et d’appliquer un effet de gravité pour faire descendre des boutons et donc combler les trous.

private boolean isRunning; définit si le programme est encore en marche ou non.

private Checker toDelete; checker contenant des boutons à supprimer.

private Semaphore antiCollider; sémaphore garantissant que le checker ayant appelé une suppression attende la fin de la suppression.

private Random rnd; utilisé pour tirer des images pour les nouveaux boutons.

private int score; score du joueur.

Outre le fait de contenir un constructeur ainsi que des getter/setter elle contient les méthodes suivantes :

public void setToDelete(Checker toDelete) attend la fin de la suppression des boutons du checker précédant puis place le checker en paramètre à supprimer.

public void run() attend la présence d’un checker contenant des boutons à supprimer, puis procède à sa suppression.

private void suppressionH() supprime des boutons d’un checker de type horizontal

private void suppressionV() supprime des boutons d’un checker de type vertical

Algorithme de fonctionnement global :



## Main

Très simple, crée un JFrame et le lance.

## VueCrush

Crée un ensemble initial de boutons, crée des threads checker horizontaux et verticaux et les lance. Crée également un objet de destruction. Après un certain temps, VueCrush arrête les threads checker et affiche le score avant de quitter.

S’occupe également de gérer l’ActionEvent lorsqu’un bouton est pressé et donc de l’échange de boutons.

# Conclusion

Cette version de CandyCrush est parfaitement fonctionnelle. La seule et grande difficulté de ce projet aura été de comprendre le fonctionnement des différents mécanismes de synchronisation offert par le langage Java. On peut cependant remettre en question l’utilité de l’utilisation de threads pour un cas comme celui-ci, Le gain apporté par l’utilisation de plusieurs threads étant en l’occurrence nul. Un programme séquentiel aurait parfaitement fait l’affaire et aurait permis d’alléger le code de tous ses éléments destinés à gérer une bonne synchronisation inter-threads. En termes d’améliorations, on pourrait rajouter des effets de mouvement ce qui améliorerait l’expérience utilisateur. Ou encore ajouter un tableau de score.