## 

## **INF-111 Travail pratique #3**

**DEUXIÈME PARTIE**

## **Automne 2014**

**Remise : mardi 11 novembre (12h00 max)**

C’est quoi ton dessin (jeu de dessin caché) ?

# Projet

Réaliser et implémenter la conception d’un jeu de dessin caché. Ce travail est nécessaire à une application de style graphique (GUI) qui sera réalisé dans le prochain travail pratique.

# Objectfs du tp2

* Mettre en application les concepts orientés-objet de classe et d’encapsulation dans un projet concret de programmation.
* Utiliser une collection de type liste implémentée avec chaînage dynamique.
* Développer des compétences transversales :
  + Utilisation de fichiers binaires entre applications.
  + Développement de méthodes de résolution de problèmes.

1. **Description un code à écrire**

Nous présumons que vous comprenez la problématique du jeu de dessin caché (piccross). Nous continuons en utilisant les classes écrites dans la partie 2 GrilleDessin, InfoBloc et ListePositionCourante).

* + 1. **Classe GrilleJeu**

**Attributs**

//Les tableaux d’indices du jeu

**private** ListePositionCourante [] tabBlocLignes;

**private** ListePositionCourante [] tabBlocColonnes;

**private** GrilleDessin dessin\_orig;

**private** GrilleDessin dessin\_cache;

**private int nbVies;**

//Évite pls appels à l'accesseur

**private** **int** taille;

Pour votre information, le dessin original est celui récupéré sur disque tandis que le dessin caché est celui présenté lors du jeu.

On utilise une liste avec position courante comme une file pour retenir les informations sur les blocs. Comme il faut une liste pour chaque ligne et une pour chaque colonne, les listes seront contenues dans deux tableaux différents. Un tableau de listes pour les lignes et un autre pour les colonnes.

Chaque liste peut être de taille différente (voir règles du jeu).

On présume que le dessin fourni au constructeur est valide alors aucune validation n'est effectuée.

* + 1. **Description des méthodes**

**Le constructeur**

Il retient la taille du dessin et le dessin original dans les attributs prévus, il alloue la mémoire pour les deux tableaux de listes et il initialise le jeu à l’aide d’une procédure locale. Finalement, il ajuste le nombre de vies par rapport à la taille (votre choix).

public GrilleJeu(GrilleDessin dessin);

**Les accesseurs**

Vous découvrirez les accesseurs à écrire au fur et à mesure que vous en aurez besoin. N’écrivez que ceux nécessaires.

Pour le bon fonctionnement de l’affichage en mode console, vous devez écrire au aussi les deux accesseurs suivants :

/\*\*

\* Retourne un tableau de tous les blocs sur une ligne

\* **@param** ligne La ligne voulue

\* **@return** Un tableau des tous les blocs sur une ligne

\*/

**public** InfoBloc[] getInfoBlocLigne(**int** ligne);

/\*\*

\* Retourne un tableau de tous les blocs sur une colonne

\* **@param** ligne La colonne voulue

\* **@return** Un tableau des tous les blocs sur une colonne

\*/

**public** InfoBloc[] getInfoBlocColonne(**int** colonne);

**Les mutateurs**

public void initialiserJeu();

Cette méthode n’est appelée qu’une seule fois par le constructeur. Elle remplit les listes des deux tableaux avec les blocs représentant les indices du jeu. Nous vous donnons l’algorithme pour les lignes et vous l’adaptez pour créer celui des colonnes (pensez sous-programmes).

Début

Pour toutes les lignes i de la grille

Initialiser la liste vide à la position i (t\_liste)

//C’est une boucle dans une boucle oui oui

Pour les colonnes j de 0 à maximum taille – 1

On doit compter le nombre de cases qui constitue un bloc (et oui une autre boucle) en commençant à la position contenue dans j

S'il y a au moins une case coloriée, on crée un info\_bloc qu’on insère dans la liste après la position courante et on déplace l’itérateur de colonne du nombre de cases + 1 pour passer au bloc suivant potentiel

Fin

Fin

**Le jeu est solutionné ou non ?**

boolean jeuEstSolutionne();

Le jeu est solutionné si les listes sont soit vides OU le nombre de cases restantes de tous les blocs est à 0. Il faut encore des boucles et nous croyons que vous devez être en mesure de trouver seul cet algorithme. Comme indice, je vous dirai d’écrire une fonction qui retourne si une liste de blocs ne contient que des 0 dans nbCasesRestantes. Il deviendra facile de vérifier toutes les listes par la suite en utilisant cette fonction.

**L’ajustement du jeu**

void ajusterJeu(int i, int j);

Ajuste le nombre de blocs restants sur la ligne i et sur la colonne j dans leur tableau respectif. Cette méthode est appelée après chaque modification au dessin caché. Nous fournissons l’algorithme pour les lignes et vous l‘adaptez pour celui des colonnes.

Début

On se positionne au début de la liste qui se trouve à la position i dans le tableau de lignes

Tant qu’on n’a pas trouvé le bloc de la ligne qui contient j

On obtient un InfoBloc de la liste (getElement)

Si la valeur de j fait partie du bloc (entre debut et debut + nb\_cases)

Décrémenter le nombre de cases restantes de 1

Remettre le bloc dans le liste (setElement \*\*\* nouveau\*\*\*\*)

On a trouvé, c’est fini

Fin Si

On passe au bloc suivant dans la liste

Fin de la boucle

Fin

1. Voici un algorithme en pseudo-code pour le programme principal :

**public** **static** **void** main(String[] args) {

GrilleDessin dessin;

Récupérer un dessin en provenance d’un fichier (ex : dessin.dsn)

**Si**  le dessin n’est pas **null**

Créer une grille de jeu à partir du dessin reçu

Si le dessin est valide selon les critère de la partie 1

Dans une boucle

UtilitaireAffichageConsole.*afficherGrilleJeu*(jeu);

Saisir le numéro de ligne entre 1 et taille

(UtilitaireValidation.lire\_Int)

Si la ligne est différente de 0

Saisir le numéro de colonne entre 1 et taille (UtilitaireValidation.lire\_Int)

Si la colonne est différente de 0

Si la case est coloriée dans le dessin original

On ajuste le jeu

Sinon

Une vie de moins

Fin si

Fin si

Tant que l’utilisateur n’a pas entré 0 pour la ligne ou qu’il n’y a plus de vie ou que le dessin est solutionné

**Sinon //!dessinValide**

Afficher un message d’erreur

Afficher un petit message de fin selon la condition d’arrêt de la boucle

Fin Si //dessin == null

Fin

**NOTEZ**

Il y a beaucoup de petits sous-programmes à écrire dans le but que vous pratiquiez. N’attendez pas à la dernière minute pour travailler, certains de ces sous-programmes demandent réflexion et doivent être redécoupé en plus petites tâches.

Ne sous-estimez pas le temps pris au débogage. Surtout si vous ne respectez pas les bonnes pratiques de programmation.

Toutes les consignes de la partie 1 s’appliquent concernant la compréhension de chaque membre de l’équipe.

Bon travail!