POLYTECH MARSEILLE Aix*Marseille Université **PEIP2**

Ingénierie Informatique

TP n°4

Affichages des bonbons échangés

Étape 21 – Fonction afficheBonbonSelectionne

La fonction `afficheBonbonSelectionne` permet de n'afficher qu'un seul bonbon, en le grisant s'il est sélectionné à la souris par le joueur. Elle prend en paramètre le `painter` qui réalisera les affichages, le tableau « modèle » `tab_jeu`, le tableau contenant les images des bonbons (décrit dans le TP n°2), les coordonnées `col` et `lig` dans le tableau « modèle » du bonbon à afficher et, enfin, un booléen `toggle_selection`. Si `toggle_selection` est égal à la valeur True, on doit dessiner le bonbon en grisé, sinon on le dessinera normalement. Pour griser une image, il suffit d'exécuter l'instruction suivante avant de faire le « *draw* » :

```
painter.setOpacity (niveau_opacité)
```

où `niveau_opacité` est un nombre réel entre 0 et 1. La valeur 0 correspond à une image totalement transparente et 1 à une image normale. En choisissant un niveau d'opacité autour de 0.3, vous devriez obtenir une image grisée comme il faut.

Si `toggle_selection` est égal à la valeur True, modifiez donc l'opacité du painter. Ensuite, quelle que soit la valeur de `toggle_selection`, affichez le bonbon. Pour tester votre fonction, cliquez sur un bonbon, sans le déplacer. Vous devriez le voir grisé. Si vous relâchez le bouton de la souris, il devrait réapparaître avec sa couleur normale.

Compléments d'informations :

Lorsque, dans une fonction, vous modifiez l'opacité, tous les affichages suivants réalisés dans la fonction seront grisés. En revanche, cette opacité sera limitée à la fonction que vous écrivez : en effet, avant chaque appel à une de vos fonctions prenant en paramètre un painter, je recrée un nouveau painter. Ainsi, les modifications que vous apportez à un painter dans une fonction ne seront pas prises en compte dans les appels ultérieurs à cette fonction ni dans des appels à d'autres fonctions.

Étape 22 – Fonction afficheSwapBonbons

La fonction `afficheSwapBonbons` prend en paramètres :
- le `painter` servant à réaliser les affichages ;
- le tableau « modèle » `tab_jeu` ;
- les coordonnées `col_s` et `lig_s` dans le tableau « modèle » du bonbon sélectionné par l'utilisateur (ce dernier a cliqué sur ce bonbon) ;
- les coordonnées `col_e` et `lig_e` dans le tableau « modèle » du bonbon avec lequel on intervertit le bonbon sélectionné ;
- un entier `deplacement` qui spécifie de combien de pixels les deux bonbons doivent se rapprocher.

Elle affiche les deux bonbons situés en (`col_s`,`lig_s`) et (`col_e`,`lig_e`) dans le tableau « modèle », après les avoir rapproché de `deplacement` pixels, comme le montre la figure 1 : à gauche, l'affichage « normal », c'est-à-dire, sans déplacement, et à droite, l'affichage par la fonction `afficheSwapBonbons` avec un `deplacement` égal à 8 pixels (le bonbon vert est ainsi déplacé de 8 pixels vers la droite et le bonbon orange de 8 pixels vers la gauche).

(a) (b)

FIGURE 1 – Affichage de l'interversion de deux bonbons.

Réécrivez la fonction `afficheSwapBonbons`. Testez votre fonction en exécutant le jeu et en essayant de déplacer un bonbon **ne faisant pas partie d'un match3** horizontalement ou verticalement. Vous devez bien voir l'animation qui déplace lesdits bonbons. Notez que la fonction `afficheSwapBonbons` ne réalise pas vraiment l'animation : elle se contente d'afficher les 2 bonbons une seule fois. L'animation sera réalisée dans une fonction ultérieure en appelant plusieurs fois votre fonction `afficheSwapBonbons` en faisant varier la valeur de `deplacement`.

Aide sur l'écriture de la fonction :

Lorsque l'utilisateur clique sur un bonbon et le déplace, ce déplacement est soit horizontal, soit vertical. Dans les deux cas, le bonbon avec lequel on l'intervertit est un bonbon voisin. Par conséquent, si l'on calcule `delta_col = col_s - col_e` et `delta_lig = lig_s - lig_e`, l'une de ces deux variables est forcément égale à 0 et l'autre à 1 ou -1.
Imaginons que `delta_col = 1`. Dans ce cas, le bouton sélectionné est à droite du bouton avec lequel on l'intervertit. Si l'on déplace le bouton sélectionné de $\alpha_x = -$`delta_col` $\times$ `deplacement` pixels vers la droite, on le rapproche de `deplacement` pixels de l'autre bonbon. De même, si l'on déplace de $-\alpha_x$ pixels l'autre bonbon, on le rapproche de `deplacement` pixels du bonbon sélectionné. On peut remarquer qu'en appliquant les déplacements α_x pour le bonbon sélectionné et $-\alpha_x$ pour l'autre, quelle que soit la valeur de `delta_col` dans $\{-1, 1\}$, on rapproche les deux bonbons (évidemment, si `delta_col = 0`, en appliquant les mêmes formules, on ne bouge pas horizontalement les bonbons). On peut appliquer le même type de formules α_y pour déplacer les bonbons verticalement.

Utilité de la fonction :

L'animation de l'interversion des deux bonbons est réalisée dans une fonction ultérieure en appelant plusieurs fois votre fonction `afficheSwapBonbons` en faisant varier la valeur de `deplacement`. Plus précisément, si on applique votre fonction avec `deplacement` = 5

pixels, si on attend 0,2 secondes puis si on rappelle votre fonction avec `deplacement` = 10 pixels, puis si on attend à nouveau 0,2 secondes et on recommence avec `deplacement` = 15 pixels, et ainsi de suite, le joueur aura réellement l'impression de voir les bonbons se déplacer l'un vers l'autre. C'est ce qui est réalisé dans la fonction `mouseReleaseEvent` dans les boucles « `for self.num_deplacement in range(1,NB_DEPLACEMENTS+1)` » : chaque `self.repaint(area)` appelle implicitement la fonction `paintEvent` qui, à son tour, appelle la vôtre.

Étape 23 – Fonction swapTableauJeu

La fonction `swapTableauJeu` prend en paramètres le tableau « modèle » `tab_jeu` ainsi que les coordonnées (`col_s`,`lig_s`) du bonbon sélectionné à la souris par l'utilisateur et les coordonnées (`col_e`,`lig_e`) du bonbon avec lequel il l'intervertit. La fonction échange les contenus des cellules correspondantes dans `tab_jeu` et ne modifie rien dans les autres cellules du tableau. Par exemple, si `tab_jeu[col_s,lig_s] = 3` et `tab_jeu[col_e,lig_e] = 1`, après exécution de la fonction, on aura `tab_jeu[col_s,lig_s] = 1` et `tab_jeu[col_e,lig_e] = 3`. Réécrivez cette fonction et testez la. Pour cela, le plus simple est d'afficher le contenu de `tab_jeu` avant et après modification (utilisez la fonction `candyPrintTableau(tab_jeu)` pour faire ces affichages plutôt qu'un `print`).

Utilité de la fonction :

La fonction `swapTableauJeu` est appelée lorsque le joueur déplace un bonbon et que cela crée un match3. Dans ce cas, les deux bonbons sont échangés dans l'espace de jeu puis l'ensemble des bonbons impliqués dans les match3 sont supprimés. Échanger les bonbons avant les suppressions est important car, si l'on intervertit horizontalement deux bonbons parce que cela crée un match3 horizontal, il se peut également qu'après interversion, les bonbons impliquent de nouveaux match3 verticaux.

Prise en compte des bonbons impliqués dans des match3

Lorsqu'un match3 a été créé par l'utilisateur, les bonbons correspondants sont détruits et ceux se trouvant au dessus descendent par gravité. Ce faisant, ils peuvent eux-mêmes induire de nouveaux match3. Les fonctions suivantes vont permettre de détecter ces match3 ainsi que la création de bonbons striés.

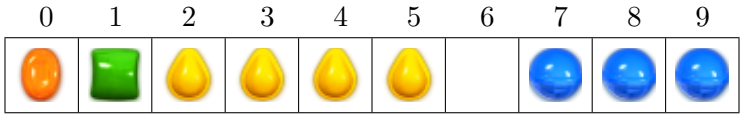
Étape 24 – Fonction calculeTabMatchHoriz

La fonction `calculeTabMatchHoriz` prend en paramètres le tableau « modèle » `tab_jeu` et un tableau d'entiers de même taille appelé `tab_match`, dont toutes les cases sont initialisées à -1. L'objectif de cette fonction est de mettre à jour toutes les cellules du tableau `tab_match` de telle sorte que ce dernier contienne le « statut » de tous les bonbons : pour toute cellule de coordonnées (`col`,`lig`) dans le tableau « modèle » `tab_jeu`, l'entier de mêmes coordonnées dans `tab_match` est égal à :

— la valeur -1 s'il y a un bonbon dans cette cellule de `tab_jeu` et qu'il ne fait pas partie d'un match3 horizontal ;

- la valeur 0 s'il n'y a pas de bonbon dans cette cellule de `tab_jeu`, ou bien s'il y a un bonbon qui fait partie d'un match3 horizontal d'exactement 3 bonbons, ou bien s'il fait partie d'un match3 horizontal de 4 bonbons ou plus mais ce n'est pas lui qui a créé ce match3 ;
- la valeur 2 s'il y a un bonbon qui fait partie d'un match3 horizontal de 4 bonbons ou plus et c'est lui qui a créé ce match3.

Pour simplifier, lorsqu'il y a des match3 de 4 bonbons ou plus, on considérera que le bonbon le plus à gauche est toujours celui qui crée ce match3. Par exemple, si on a la ligne suivante du tableau `tab_jeu` :



alors, les bonbons des colonnes 0 et 1 ne font partie d'aucun match3. Donc leurs valeurs dans le tableau `tab_match` seront égales à -1. Ensuite, on a un match3 de 4 bonbons jaunes. Celui le plus à gauche aura donc la valeur 2 dans `tab_match` et les trois à sa droite auront la valeur 0. La colonne 6 ne contient pas de bonbon, la valeur correspondante dans `tab_match` sera donc 0. Enfin, il y a un match3 de seulement 3 bonbons bleus. Ils auront tous la valeur 0 dans `tab_match`. À la fin de l'exécution de votre fonction `calculeTabMatchHoriz`, on obtiendra le tableau `tab_match` suivant :

0	1	2	3	4	5	6	7	8	9
-1	-1	2	0	0	0	0	0	0	0

Réécrivez la fonction `calculeTabMatchHoriz`. Pour la tester, exécutez le jeu et réalisez des **match3 horizontaux** de 3 bonbons ou plus. Vous devez voir les bonbons se supprimer et, si le match3 contient 4 bonbons ou plus, vous devez voir apparaître un bonbon strié.

Aide sur l'écriture de la fonction :

L'algorithme pour mettre à jour le tableau `tab_match` avec les règles décrites ci-dessus est le suivant : on boucle sur les lignes et les colonnes pour parcourir tout l'espace de jeu, en se déplaçant de la gauche vers la droite et du bas vers le haut.
Pour chaque couple (`col`,`lig`), on teste si `tab_jeu[col,lig]` contient un bonbon ou non. S'il n'en contient pas, on doit mettre 0 dans la case correspondante de `tab_match`. Sinon, il faudrait tester si le bonbon fait partie d'un match3 et, le cas échéant, lui affecter la valeur 0 s'il y a 3 bonbons dans le match3 et 2 s'il y en a plus. Le problème, si on réalise précisément cela, c'est que l'on aura des 2 dans tous les bonbons des match3 impliquant au moins 4 bonbons (or, ici, on veut un bonbon avec un 2 et les autres avec des 0). Pour pallier cela, on procède légèrement différemment :
S'il n'y a pas de bonbon situé en (`col`,`lig`) dans `tab_jeu`, on met 0 dans la case correspondante de `tab_match`. Sinon, si `tab_match[col,lig]` = -1, alors on appelle la fonction `nbBonbonsContigusHoriz` pour déterminer le nombre de bonbons contigus au bonbon situé en (`col`,`lig`) et de même type que lui. Si ce nombre est supérieur ou égal à 3, il y a un match3 et on met dans `tab_match` des 0 à **tous les bonbons impliqués dans le match3**. Ensuite, si le nombre est supérieur ou égal à 4, on remet à jour `tab_match[col,lig]` en lui affectant la valeur 2.
En procédant ainsi, on garantit qu'un seul des bonbons aura la valeur 2 et les autres la valeur 0. En effet, la première fois qu'on rencontre un bonbon impliqué dans un match3 d'au moins 4 bonbons, on lui affecte la valeur 2 et on affecte des 0 à tous les autres bonbons de ce match3. Quand votre parcours de l'espace de jeu vous amène à un de ces autres bonbons, ils n'ont plus la valeur -1 et, par conséquent, on ne va pas appeler la fonction `nbBonbonsContigusHoriz` ni,

a fortiori, essayer de modifier le tableau `tab_match`.
Astuce : il est plus facile de créer des match3 de 4 bonbons ou plus dans des espaces de jeu de grande taille. Si vous doublez les valeurs des constantes `NB_COLONNES_JEU` et `NB_LIGNES_JEU` en début de fichier, vous obtiendrez un plateau de jeu suffisamment grand pour avoir plein d'opportunités de créer de tels match3.

Utilité de la fonction :
À la fin de l'exécution de votre fonction, le tableau `tab_match` contient des 0 aux emplacements de `tab_jeu` où les bonbons doivent être supprimés (parce qu'ils font partie de match3). Il contient des 2 aux emplacements où l'on doit créer des bonbons striés. Enfin, les cellules avec des -1 n'engendrent aucune modification du tableau `tab_jeu`.
Chaque fois que l'utilisateur déplace un bonbon et crée ainsi un match3, une fonction appelée `calculeTabMatch` est exécutée. Celle-ci va mettre à jour le tableau `tab_match` en appelant votre fonction `calculeTabMatchHoriz` pour tenir compte des match3 horizontaux, et la fonction `calculeTabMatchVert` pour tenir compte des match3 verticaux (cf. l'étape suivante). À l'issue de ces appels, on sait précisément comment le tableau de jeu `tab_jeu` doit être mis à jour pour tenir compte des nouveaux match3. Une autre fonction réalisera ces changements.

Étape 25 – Fonction calculeTabMatchVert

La fonction `calculeTabMatchVert` est similaire à la fonction `calculeTabMatchHoriz`, excepté que l'on examine les match3 verticaux et non les match3 horizontaux. Réécrivez cette fonction et testez la. Ici, quand 4 bonbons ou plus sont impliqués dans un match3, c'est celui le plus bas qui doit avoir la valeur 2 dans `tab_match`, les autres prenant la valeur 0.