POLYTECH MARSEILLE Aix*Marseille Université **PEIP2**

Ingénierie Informatique

TP n°5

Prise en compte des match3 et des stries

Étape 26 – Fonction calculeTabMatch

La fonction `calculeTabMatch` prend en paramètres deux tableaux : `tab_jeu`, le tableau « modèle », et `tab_match`, le tableau qui recense l'état des bonbons (cf. les fonctions `calculeTabMatchHoriz` et `calculeTabMatchVert` du TP n°4). Son but est de « fusionner » les informations fournies par ces fonctions. Pour cela, elle commence par créer 2 nouveaux tableaux, appelons-les `tab_match_horiz` et `tab_match_vert`, de même taille que `tab_match` et initialisés avec des -1. Elle appelle ensuite la fonction `calculeTabMatchHoriz` en lui passant en argument le tableau `tab_match_horiz` et la fonction `calculeTabMatchVert` en lui passant en argument le tableau `tab_match_vert`. Il ne reste plus qu'à fusionner les informations de ces deux tableaux. Pour cela, il suffit, pour chaque cellule (`col`,`lig`), d'affecter à `tab_match[col,lig]` le maximum de `tab_match_horiz[col,lig]` et `tab_match_vert[col,lig]`. Réécrivez la fonction `calculeTabMatch` et testez la en réalisant des match3. Vous devez voir les bonbons impliqués dans les match3, et seulement ceux-ci, se supprimer correctement et les bonbons créant des match3 d'au moins 4 bonbons doivent devenir striés. Vous pouvez afficher via la fonction `candyPrintTableau` le contenu du tableau `tab_match` que vous avez calculé afin de bien vérifier que votre code est correct.

Compléments d'informations :

Rappel : les fonctions `calculeTabMatchHoriz` et `calculeTabMatchVert` du TP précédent prennent en paramètres un tableau `tab_jeu` et un tableau `tab_match` initialisé avec des -1, et elles modifient ce dernier de sorte que, dans chaque cellule, on ait :

- la valeur -1 s'il y a un bonbon dans la cellule correspondante de `tab_jeu` et qu'il ne fait pas partie d'un match3 horizontal/vertical ;
- 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/vertical d'exactement 3 bonbons, ou bien s'il fait partie d'un match3 horizontal/vertical 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/vertical de 4 bonbons ou plus et c'est lui qui a créé ce match3.

Pourquoi calculer le max de `tab_match_horiz[col,lig]` et `tab_match_vert[col,lig]` ? Imaginons que que le premier ait la valeur 2 (création d'un bonbon strié) et l'autre la valeur 0 (destruction du bonbon parce qu'il fait partie d'un match3). Le bonbon fait donc partie de 2 match3, l'un horizontal et l'autre vertical. Il faut conserver le bonbon strié à cause du match3

horizontal, donc la valeur $2 = \max\{2, 0\}$. De même si `tab_match_horiz[col,lig]` = 2 et `tab_match_vert[col,lig]` = -1, on a trouvé un match3 de 4 bonbons ou plus horizontalement, mais pas de match3 verticalement. Il faut conserver le fait qu'on crée un bonbon strié à cause du match3 horizontal. Enfin, si `tab_match_horiz[col,lig]` = 0 et `tab_match_vert[col,lig]` = -1, il y a un match3 de 3 bonbons horizontalement mais pas de match3 vertical. Il faut tenir compte du match3 horizontal et supprimer le bonbon, donc garder la valeur $0 = \max\{0, -1\}$. Pourquoi créer les deux tableaux `tab_match_horiz` et `tab_match_vert` plutôt que d'exécuter directement les fonctions `calculeTabMatchHoriz` et `calculeTabMatchVert` en leur passant en argument `tab_match` ? Le problème, ici, c'est que certains bonbons peuvent faire partie à la fois d'un match3 horizontal et d'un match3 vertical. Supposons que le bonbon situé en (`col`,`lig`) fasse partie d'un match3 horizontal de 3 bonbons et d'un match3 vertical de 4 bonbons qu'il a lui-même créé. Il doit donc avoir la valeur 2 dans `tab_match`. En appelant `calculeTabMatchHoriz` en lui passant en argument `tab_match`, on aura `tab_match[col,lig]` = 0. Puis, en appelant `calculeTabMatchVert`, toujours en lui passant en argument `tab_match`, on conservera cette valeur car, comme au début de l'exécution de la fonction, `tab_match[col,lig]` n'est pas égal à -1, votre fonction `calculeTabMatchVert` ne mettra pas à jour la valeur de `tab_match[col,lig]`. Le résultat sera donc erroné. C'est pourquoi, il faut appeler `calculeTabMatchHoriz` et `calculeTabMatchVert` avec deux tableaux différents initialisés avec des -1.

Utilité de la fonction :

Après exécution de la fonction `calculeTabMatch`, chaque cellule du tableau `tab_match` a :

- la valeur -1 s'il y a un bonbon dans la cellule correspondante de `tab_jeu` et qu'il ne fait pas partie d'aucun match3, horizontal ou vertical ;
- 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 ou vertical) et, si celui-ci contient 4 bonbons ou plus, ce n'est pas ce bonbon qui l'a créé ;
- la valeur 2 s'il y a un bonbon dans la cellule correspondante de `tab_jeu` qui fait partie d'un match3 de 4 bonbons ou plus et c'est lui qui a créé ce match3.

Chaque fois que l'utilisateur déplace un bonbon et qu'on identifie que cela crée au moins un match3, on appelle la fonction `calculeTabMatch` afin de déterminer les modifications à réaliser dans l'espace de jeu (le tableau `tab_jeu`) : on sait qu'il faudra supprimer les bonbons dans les cases où `tab_match` vaut 0 et strier les bonbons dans les cases où `tab_match` vaut 2.

Étape 27 – Fonction updateTabMatchAvecStrie

Cette fonction prend en paramètres les deux mêmes tableaux `tab_jeu` et `tab_match` que la fonction précédente. Elle parcourt toutes les cellules de `tab_jeu`. Pour chaque cellule de coordonnées (`col`,`lig`), si le bonbon `tab_jeu[col,lig]` a un supertype égal à `SUPERTYPE_BONBON_STRIE_HORIZ` et si `tab_match[col,lig]` est supérieur ou égal à 0, elle remplace tous les -1 se situant sur les cellules de la ligne `lig` de `tab_match` par des 0. De même, si le bonbon `tab_jeu[col,lig]` a un supertype égal à `SUPERTYPE_BONBON_STRIE_VERT` et si `tab_match[col,lig]` est supérieur ou égal à 0, elle remplace tous les -1 se situant sur les cellules de la colonne `col` de `tab_match` par des 0. Réécrivez la fonction `updateTabMatchAvecStrie` et testez la en réalisant des match3 impliquant des bonbons striés.

Utilité de la fonction :

Le but de cette fonction est de mettre à jour `tab_match` de telle sorte que les bonbons situés sur la même ligne (resp. la même colonne) qu'un bonbon strié horizontalement (resp. verticalement) impliqué dans un nouveau match3 ait une valeur supérieure ou égale à 0 dans `tab_match`. Ainsi, ils seront automatiquement détruits. Pourquoi ne remplacer que les -1 par des 0 ? Tout simplement par ce que si, sur la ligne (resp. la colonne) en question, il y a un 2 (donc un bonbon qui doit devenir strié), on ne veut pas supprimer ce bonbon, on veut le strier pour la suite du jeu.

Étape 28 – Fonction ajouteUneStrie

La fonction `ajouteUneStrie` prend en paramètres le tableau « modèle » `tab_jeu`, les coordonnées (`col`,`lig`) d'un bonbon de ce tableau et une direction de déplacement (`DIRECTION_DROITE`, `DIRECTION_GAUCHE`, *etc.*). Elle remplace dans `tab_jeu` ce bonbon par un nouveau bonbon du même type mais strié horizontalement si la direction est `DIRECTION_DROITE` ou `DIRECTION_GAUCHE`, ou verticalement si la direction est `DIRECTION_HAUT` ou `DIRECTION_BAS`. Réécrivez la fonction `ajouteUneStrie` et testez la en créant des match3 impliquant au moins 4 bonbons. Vous devez voir les bonbons striés résultant de ces match3. Notez bien que, dans les règles de candy crush, la direction des stries correspond à la direction de déplacement de la souris et non au fait qu'on crée un match3 horizontal ou vertical.

Utilité de la fonction :

Lorsqu'un bonbon a été déplacé (soit par l'utilisateur, soit en descendant par gravité) et a ainsi crée un **nouveau match3 de 4 bonbons ou plus**, il ne doit pas être supprimé mais il doit être transformé en bonbon strié (c'est l'une des règles de candy crush). Le but de la fonction `ajouteUneStrie` est de réaliser cette opération. Elle est appelée par la fonction de l'étape suivante, qui rajoute toutes les stries créées par le déplacement de l'utilisateur.

Étape 29 – Fonction ajouteBonbonsStries

La fonction `ajouteBonbonsStries` prend en paramètres les sempiternels tableaux `tab_jeu` et `tab_match`, ainsi qu'une direction de déplacement (`DIRECTION_DROITE`, `DIRECTION_GAUCHE`, *etc.*). Elle parcourt tous les bonbons du tableau `tab_match`. Chaque fois qu'elle en rencontre un dont la valeur est 2, elle le strie dans `tab_jeu` selon la direction de déplacement passée en paramètre. Réécrivez la fonction `ajouteBonbonsStries` et testez la en créant des match3 impliquant au moins 4 bonbons. Vous devez voir les bonbons striés résultant de ces match3.

Aide sur l'écriture de la fonction :

Vous pourrez avantageusement vous servir de votre fonction `ajouteUneStrie` pour strier les bonbons.

Utilité de la fonction :

Comme nous l'avons vu, lorsqu'un bonbon a été déplacé (soit par l'utilisateur, soit en descendant par gravité) et a ainsi créé un nouveau match3 de 4 bonbons ou plus, il ne doit pas être supprimé mais il doit être transformé en bonbon strié. C'est précisément cette information qui

est encodée par les valeurs 2 dans `tab_match`.
La fonction est appelée `ajouteBonbonsStries` par la fonction `mouseReleaseEvent` de l'interface de jeu, qui réalise toutes les opérations nécessaires lorsque le joueur relâche la souris. Dans ce cas, on commence par calculer `tab_match` et on en déduit quels bonbons doivent être striés pour la suite du jeu.