POLYTECH MARSEILLE Aix*Marseille Université **PEIP2**

Ingénierie Informatique

TP n°2

Tests de match3

Étape 9 – Fonction nbBonbonsContigusHoriz

La fonction `nbBonbonsContigusHoriz` prend en premier paramètre un tableau numpy `tab_jeu` de `NB_COLONNES_JEU` colonnes et `NB_LIGNES_JEU` lignes contenant les bonbons affichés par le jeu, encodés par des entiers (c'est ce que l'on appelle le tableau « modèle »). Ses deux autres paramètres sont un numéro de colonne `col` et un numéro de ligne `lig` du tableau. La fonction renvoie le nombre de bonbons situés sur la même ligne, côte à côte les uns des autres et, notamment, du bonbon placé en (`col`, `lig`) et ayant le même type que ce dernier (Attention : ils ont le même type, mais pas forcément le même supertype). Par exemple, sur la figure 1, la fonction appliquée aux coordonnées de n'importe quel bonbon vert de la partie gauche, renverra le nombre 3 car on a bien 3 bonbons verts situés côte à côte. En revanche, dans la partie droite, la fonction, appliquée aux coordonnées du bonbon strié, renverra 1 car, entre lui et les 2 autres bonbons verts, est intercalé un bonbon rouge (donc ils ne sont pas contigus). Enfin, la fonction, appliquée aux coordonnées de n'importe lequel des deux bonbons à droite du bonbon rouge, renverra 2 car il y a seulement deux bonbons consécutifs de couleur verte.

FIGURE 1 – À gauche : 3 bonbons verts contigus. À droite : 1 bonbon vert contigü à gauche du bonbon rouge et 2 bonbons verts contigus à droite du bonbon rouge.

Réécrivez la fonction `nbBonbonsContigusHoriz`. Testez votre fonction. Pour cela, il suffit d'exécuter le jeu. Si vous n'observez pas les bonbons, vous avez sans doute une boucle infinie. Si vous déplacez un bonbon dont vous voyez sur l'écran qu'il va créer un match3 horizontal, et que les bonbons impliqués ne se détruisent pas (ou inversement), c'est que vous ne calculez pas correctement le nombre de bonbons contigus de même type. Si le jeu a l'air de fonctionner correctement, votre fonction est très probablement correcte.

Aide sur l'écriture de la fonction :

L'algorithme pour calculer le nombre de bonbons contigus de même type n'est pas très compliqué : créez une variable `somme` qui comptera ce nombre de bonbons. Initialisez `somme` à 1 pour tenir compte du bonbon situé en `tab_jeu[col,lig]`. Ensuite, parcourez les bonbons de coordonnées (`ncol`,`lig`) à la gauche du bonbon (`col`,`lig`) en vous éloignant de plus en plus de `col` (donc en regardant d'abord le bonbon en `ncol` = `col-1`, puis en `ncol` = `col-2`, *etc.*) :

tant que vous restez dans le tableau de jeu (donc `ncol` ≥ 0) et que le **type** du bonbon en (`ncol`,`lig`) est égal à celui du bonbon situé en (`col`,`lig`), incrémentez la variable `somme` de 1 (incrémenter une variable = augmenter sa valeur de 1).
Idée : en anglais, « tant que » se traduit par `while`.
Astuce : vous pourrez avantageusement utiliser la fonction `getTypeBonbon` que vous avez réécrite précédemment.
Faites de même avec les bonbons à droite du bonbon situé en (`col`,`lig`), en prenant garde, là aussi, de bien rester dans le tableau de jeu, c'est-à-dire `ncol` < `NB_COLONNES_JEU`. Après avoir exécuté ces deux boucles, l'une après l'autre, dans la variable `somme`, vous avez le nombre de bonbons situés côte à côte de même *type* que le bonbon situé en (`col`,`lig`). Pour achever l'écriture de la fonction, il vous suffit donc de renvoyer la valeur de cette variable.
Pour tester votre fonction, si vous exécutez le jeu et que vous n'observez pas les bonbons, vous avez sans doute une boucle infinie et cela signifie sans doute que vous avez oublié de modifier la valeur de `ncol` dans au moins l'une de vos 2 boucles. Si vous déplacez un bonbon dont vous voyez sur l'écran qu'il va créer un match3 horizontal, et que les bonbons impliqués ne se détruisent pas, c'est que le calcul de la somme est incorrect ou bien que vous ne renvoyez pas la valeur de la variable `somme`. Si le jeu a l'air de fonctionner correctement, votre fonction est très probablement correcte.

Utilité de la fonction :

La fonction `nbBonbonsContigusHoriz` est appelée chaque fois que l'on a besoin de tester si un bonbon induit un match3 horizontal. Par exemple, on en a besoin dans la fonction `initialiseTableauJeu` qui initialise le jeu. En effet, on ne veut pas débuter avec une configuration où des bonbons forment déjà des match3 puisque ceux-ci devraient être supprimés (du fait qu'ils font partie d'un match3). On s'en sert également chaque fois que l'utilisateur déplace un bonbon pour déterminer si cela crée ou non un match3 horizontal.

Étape 10 – Fonction nbBonbonsContigusVert

La fonction `nbBonbonsContigusVert` est similaire à `nbBonbonsContigusHoriz` à ceci près qu'elle compte le nombre de bonbons situés côte à côte **verticalement**, donc sur une colonne, plutôt que sur une ligne. Elle a les mêmes paramètres que la fonction `nbBonbonsContigusHoriz`. Copiez-collez le code de `nbBonbonsContigusHoriz` dans la fonction `nbBonbonsContigusVert` et adaptez ce code de manière à ce que ce soient les lignes `lig` qui varient et non les colonnes `col`. On rappelle que le nombre de lignes dans le tableau est `NB_LIGNES_JEU` et non `NB_COLONNES_JEU`. Testez votre fonction de la même manière que vous aviez testé la fonction `nbBonbonsContigusHoriz`.

Étape 11 – Fonction match3

La fonction `match3` a les mêmes paramètres que `nbBonbonsContigusHoriz` et `nbBonbonsContigusVert`. Elle renvoie le booléen `True` si la case `tab_jeu[col,lig]` du tableau de jeu est impliquée dans un « match3 » horizontal **ou** vertical.

Réécrivez la fonction `match3`. Testez-la de la même manière que vous aviez testé les deux fonctions `nbBonbonsContigusHoriz` et `nbBonbonsContigusVert`.

Aide sur l'écriture de la fonction :

Pour écrire la fonction `match3`, ne copiez-collez pas les codes que vous avez écrits dans les fonctions `nbBonbonsContigusHoriz` et `nbBonbonsContigusVert`. Plutôt, faites des appels à ces deux fonctions.

Initialisation du jeu

Étape 12 – Fonction initialiseTableauVide

La fonction `initialiseTableauVide` prend en paramètre le tableau modèle `tab_jeu` de `NB_COLONNES_JEU` colonnes et `NB_LIGNES_JEU` lignes contenant les bonbons encodés par des entiers. Elle place dans chacune des cellules de ce tableau le nombre -1 (-1 aura pour signification : pas de bonbon dans la cellule). Pour réécrire cette fonction, il faut donc parcourir toutes les lignes du tableau et, pour chaque ligne, parcourir toutes ses colonnes (n'oubliez pas que, dans un tableau, les indices de lignes et de colonnes commencent à 0).

Aide sur l'écriture de la fonction :

La fonction `initialiseTableauVide` est appelée par la fonction `initialiseTableauJeu`. Celle du package candyProf affiche dans la console Python la valeur de `tab_jeu` avant et après appel à votre fonction. Vérifiez bien que, dans la console, `tab_jeu` ne contient que des -1 après exécution de votre fonction.

Utilité de la fonction :

La fonction `initialiseTableauVide` n'est pas essentielle pour le jeu mais elle est utile pour accélérer l'exécution de la fonction `initialiseTableauJeu`, qui remplit le tableau de jeu lorsque l'on démarre une partie. En effet, dans cette dernière, pour chaque case du tableau, on doit choisir aléatoirement un bonbon et le placer. Imaginons que ce bonbon soit un carré vert. Si l'espace de jeu contient déjà deux carrés verts à sa droite, cela produit un match3, ce que l'on ne souhaite pas avoir lorsqu'on initialise le jeu. Par conséquent, on est obligé de choisir un autre bonbon pour éviter ce match3. À l'issue de l'exécution de la fonction `initialiseTableauVide`, on a l'assurance que les deux bonbons situés à droite de notre carré vert ne peuvent être eux-mêmes des carrés verts car ils ont la valeur -1. On a donc potentiellement moins de match3 et, par conséquent, la fonction `initialiseTableauJeu` s'exécutera plus vite.

Étape 13 – Fonction initialiseTableauJeu – 1ère version (provisoire)

La fonction `initialiseTableauJeu` prend en paramètre exactement le même tableau que la fonction `initialiseTableauVide`. Elle initialise ce tableau en appelant `initialiseTableauVide`. Puis elle remplit toutes les cellules du tableau de manière aléatoire : pour cela, elle exploite la fonction `randint` du module `random` qui permet de tirer au hasard des nombres entiers. Ainsi, tout appel à `random.randint(a,b)` vous retournera un nouvel entier `n` tiré « aléatoirement » et tel que $\texttt{a} \leq \texttt{n} \leq \texttt{b}$. La fonction `randint` vous permettra ainsi de facilement générer aléatoirement les *types* des bonbons.

Notez que ces types doivent être des nombres entiers compris entre 0 et `NB_TYPES_BONBONS - 1`, comme indiqué dans la vidéo « principes de programmation du jeu ». En début de jeu, les bonbons sont tous normaux (non striés), donc leur supertype est forcément égal à `SUPERTYPE_BONBON_NORMAL`. Étant donné un type et un supertype, vous pourrez avantageusement utiliser votre fonction `getBonbon` pour obtenir le bonbon correspondant. Réécrivez la fonction `initialiseTableauJeu`.

Pour la tester, exécutez votre jeu et regardez si les bonbons affichés en début de partie ont l'air corrects ou non. Dans cette première version de la fonction, on peut avoir des match3, ce n'est pas grave, vous réglerez ce problème à l'étape suivante. Pour l'instant, ce qui est important, c'est d'avoir des bonbons dans tout l'espace de jeu.

Utilité de la fonction :

La fonction `initialiseTableauJeu` est appelée chaque fois que l'on démarre une partie. Elle sert à remplir de bonbons le tableau de jeu.

Étape 14 – Fonction initialiseTableauJeu – version finale

Modifiez votre fonction de telle sorte qu'aucun bonbon de votre espace de jeu ne forme de match3. Testez en cliquant plusieurs fois sur le bouton « rejouer » (cela forcera le programme à réinitialiser l'espace de jeu et, donc, à appeler votre fonction.

Aide sur l'écriture de la fonction :

L'idée est la suivante : pour chaque cellule de votre tableau, après avoir tiré aléatoirement le bonbon de la cellule, vérifiez que celui-ci ne forme pas de match3 (en exploitant votre fonction `match3`). Si ce n'est pas le cas, retirez aléatoirement un nouveau bonbon pour la cellule, puis revérifiez qu'il ne forme pas de match3, *etc.* On boucle donc sur ces opérations tant qu'on n'a pas réussi à trouver un bonbon qui ne forme pas de match3.