POLYTECH MARSEILLE Aix*Marseille Université **PEIP2**

Ingénierie Informatique

TP n°3

Affichage du tableau de jeu

Étape 15 – Fonction afficheTableauJeu – affichage de la bordure

La fonction `afficheTableauJeu` permet d'afficher à l'écran les bonbons de l'espace de jeu ainsi que la bordure autour de ces bonbons. Elle prend en paramètres :

1. `painter` : le « painter » de Qt qui réalisera effectivement les affichages (cf. la vidéo « principes de programmation du jeu ») ;
2. `tab_jeu` : le tableau « modèle » contenant le bonbons, encodés sous forme d'entiers ;
3. `images` : un tableau 2D contenant les images des bonbons. Ainsi `images[s,t]` est l'image du bonbon dont le supertype est `s` et dont le type est `t`.

En utilisant `painter.drawRect`, affichez la bordure autour des bonbons. On rappelle que, dans `tab_jeu`, il y a `NB_LIGNES_JEU` lignes et `NB_COLONNES_JEU` colonnes. Par ailleurs, les paramètres de la « méthode » `drawRect` du `painter` sont :

1. l'abscisse `x` en pixels du coin supérieur gauche du rectangle ;
2. l'ordonnée `y` en pixels du coin supérieur gauche du rectangle ;
3. la largeur en pixels du rectangle ;
4. la hauteur en pixels du rectangle.

Testez votre fonction en exécutant le jeu : vous devez voir votre bordure au bon endroit et aucun bonbon.

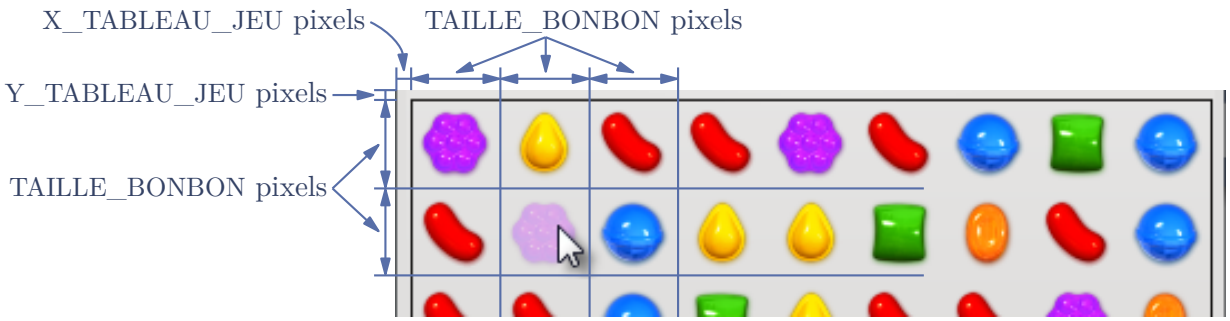

FIGURE 1 – Les tailles en pixels de l'espace de jeu.

Étape 16 – Fonction afficheTableauJeu – version finale

Après l'affichage de la bordure, on affiche les bonbons. Pour cela, on parcourt toutes les lignes et toutes les colonnes de `tab_jeu`. Pour chaque cellule, on extrait le type et le supertype du bonbon (en exploitant les fonctions du TP n°1). Cela permet de déterminer l'image du bonbon (cf. l'explication de la fonction précédente). Puis on appelle `painter.drawPixmap`, comme indiqué dans la vidéo « principes de programmation du jeu ». On rappelle que les paramètres de la « méthode » `drawPixmap` de `painter` sont :

1. l'abscisse `x` en pixels du coin supérieur gauche de l'image ;
2. l'ordonnée `y` en pixels du coin supérieur gauche de l'image ;
3. l'image elle-même.

Aide sur l'écriture de la fonction :

Pour obtenir les valeurs des coordonnées des coins supérieurs gauches des images des bonbons, vous pourrez avantageusement exploiter les fonctions `getXPixel` et `getYPixel` que vous avez écrites dans le TP n°1.
Si vous relisez votre code des fonctions `getXPixel` et `getYPixel`, vous pourrez facilement déduire que les coordonnées du coin supérieur gauche de l'image du bonbon le plus haut à gauche de la fenêtre de jeu sont (`X_TABLEAU_JEU`,`Y_TABLEAU_JEU`). Donc si les coordonnées du coin supérieur gauche de votre bordure sont également (`X_TABLEAU_JEU`,`Y_TABLEAU_JEU`), en affichant les bonbons, vous allez écrire par dessus la bordure, qui risque donc de disparaître. Pour vérifier que votre bordure est correcte, tentez de déplacez des bonbons qui ne forment pas de match3 sur la ligne la plus haute, celle la plus basse, la colonne la plus à gauche et celle la plus à droite. Si votre bordure disparaît, c'est que les paramètres du `drawRect` de l'étape précédente étaient incorrects.

Calculs des bonbons échangés

Étape 17 – Fonction getDirection

Le fonction `getDirection` prend en 1er paramètre un événement `event` généré par Qt, qui représente un déplacement de la souris. Ses deux autres paramètres sont la colonne `col` et la ligne `lig` d'un bonbon dans le tableau « modèle ». Le but de la fonction est de renvoyer la direction dans laquelle le bonbon a été déplacé (s'il a été déplacé). La fonction commence donc par extraire les coordonnées de la souris en pixels :

- `event.pos().x()` est l'abscisse en pixels de la souris ;
- `event.pos().y()` est l'ordonnée en pixels de la souris.

Ensuite, elle transforme ces coordonnées pixels en coordonnées du « modèle » (colonne,ligne). Appelons ces nouvelles coordonnées « modèle » `col_souris` et `lig_souris`. La fonction compare alors `(col_souris,lig_souris)` à `(col,lig)` et renvoie la direction correspondante :

- si `lig_souris` = `lig` et `col_souris` < `col` et `col` > `0` :
  alors la fonction renvoie la valeur `DIRECTION_GAUCHE` (c'est une « constante » définie vers le début du fichier `candy.py`) ;

- si `lig_souris` = `lig` et `col_souris` > `col` et `col` < `NB_COLONNES_JEU - 1` :
  alors la fonction renvoie la valeur `DIRECTION_DROITE` ;
- si `col_souris` = `col` et `lig_souris` < `lig` et `lig` > `0` :
  alors la fonction renvoie la valeur `DIRECTION_HAUT` ;
- si `col_souris` = `col` et `lig_souris` > `lig` et `lig` < `NB_LIGNES_JEU - 1` :
  alors la fonction renvoie la valeur `DIRECTION_BAS` ;
- si l'on n'est dans aucun des cas précédents :
  alors la fonction renvoie `DIRECTION_AUCUNE` (pour indiquer que l'on ne déplace pas le bonbon).

Réécrivez la fonction `getDirection`. Testez la en exécutant le jeu et en déplaçant à la souris des bonbons dans toutes les directions. Le plus simple pour vérifier que votre fonction est correcte est sans doute d'afficher via un `print` la valeur de la direction que vous renvoyez avant de faire le `return`.

Utilité de la fonction :

Il est fastidieux d'utiliser des coordonnées en pixels pour déterminer le mouvement (haut, bas, droite, gauche) qu'a effectué le joueur. La fonction `getDirection` permet de s'affranchir de ces pixels et de ne garder que l'information sur la direction (haut, bas, droite, gauche) du mouvement.
Vous noterez l'intérêt des dernières conditions de chaque test (`col > 0`, `col < NB_COLONNES_JEU - 1`, *etc.*) : elles servent à éviter de bouger en dehors du plateau de jeu les bonbons qui se trouvent sur des bords.

Étape 18 – Fonction getColonneBonbonEchange

La fonction `getColonneBonbonEchange` prend en 1er paramètre la colonne `col` d'un bonbon dans le tableau « modèle » et en 2ème paramètre une direction de déplacement telle que renvoyée par la fonction `getDirection`. Elle renvoie la colonne sur laquelle se trouvera le bonbon après le déplacement (donc `col + 1` s'il s'agit d'un déplacement vers la droite, `col - 1` s'il s'agit d'un déplacement vers la gauche et `col` sinon). Réécrivez la fonction `getColonneBonbonEchange` et testez la. Comme dans la fonction précédente, déplacez des bonbons pour tester et affichez via un `print` la coordonnée `col` ainsi que la coordonnée que vous retournez.

Utilité de la fonction :

Les fonctions `getColonneBonbonEchange` et `getLigneBonbonEchange` permettent de déterminer quel est, dans le tableau modèle `tab_jeu`, le bonbon à échanger avec le bonbon déplacé à la souris par l'utilisateur. Ces fonctions servent donc quand l'utilisateur déplace des bonbons. Notamment, elles sont utilisées dans la fonction `mouseReleaseEvent` de l'interface de jeu.

Étape 19 – Fonction getLigneBonbonEchange

La fonction `getLigneBonbonEchange` est similaire à la fonction précédente à ceci près que son premier paramètre est un numéro de ligne et qu'elle renvoie la ligne où se situe le bonbon après déplacement. Réécrivez la fonction `getLigneBonbonEchange` et testez la.

Étape 20 – Fonction getRegion

La fonction `getRegion` prend en paramètres les coordonnées `col0`, `lig0` d'un bonbon dans le tableau « modèle » (donc exprimées en colonnes et lignes), ainsi que les coordonnées `col1`, `lig1` d'un autre bonbon dans le tableau « modèle ». La fonction calcule les coordonnées **en pixels** (`x_pixel,y_pixel`) du coin supérieur gauche ainsi que la largeur `w` et la hauteur `h` en pixels de la plus petite région englobant les deux bonbons. Ce sera la région dont l'affichage doit être rafraîchi quand on échange les deux bonbons. Sur la figure 2, vous pouvez voir quelques exemples de cette région englobante. Pour retourner la région englobante, la fonction renvoie un rectangle Qt :

```
return QtCore.QRect(x_pixel,y_pixel,w,h)
```

où (`x_pixel,y_pixel`) sont les coordonnées du coin supérieur gauche de la région, `w` est la largeur de la région et `h` est sa hauteur (en pixels). On rappelle que la hauteur et la largeur d'un bonbon sont de `TAILLE_BONBON` pixels. Réécrivez la fonction `getRegion`. Pour la tester, déplacez à la souris dans n'importe quelle direction des bonbons qui ne forment pas de match3. Vous devez voir l'animation des bonbons qui s'échangent puis reviennent à leur position initiale. Si ce n'est pas le cas, c'est que votre fonction est incorrecte.

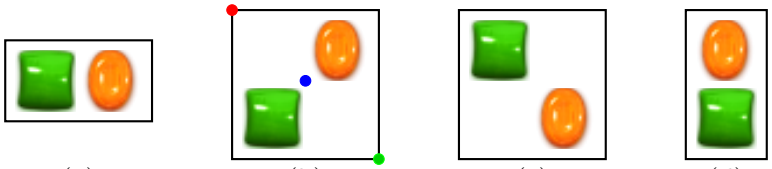

FIGURE 2 – Exemples de régions englobantes symbolisées par des bordures.

Aide sur l'écriture de la fonction :

Astuce : pour calculer la région, commencez par calculer le min et le max de $\{\texttt{col0}, \texttt{col1}\}$ ainsi que de $\{\texttt{lig0}, \texttt{lig1}\}$ et, par la suite, ne manipulez que ces min et max.

Prenons comme exemple la figure 2(b) et supposons que le bonbon vert a pour coordonnées $(\texttt{col0}, \texttt{lig0})$ et le bonbon orange les coordonnées $(\texttt{col1}, \texttt{lig1})$. Puisque le bonbon orange est une colonne à droite du vert, $\texttt{col1} = \texttt{col0} + 1$. De même, puisque le bonbon orange est 1 ligne au dessus du vert, $\texttt{lig1} = \texttt{lig0} - 1$ (on se rappelle que les ordonnées augmentent en se déplaçant vers le bas). Par conséquent, le min de $\{\texttt{col0}, \texttt{col1}\}$ est égal à `col0` et le min de $\{\texttt{lig0}, \texttt{lig1}\}$ est égal à `lig1`. Cela correspond donc à la colonne du bonbon vert et la ligne du bonbon orange. Ce sont les coordonnées du bonbon qui serait situé entre le point rouge et le point bleu, donc le bonbon le plus haut à gauche dans la région englobante (symbolisée par le rectangle noir). Si on prend les coordonnées en pixels de ce bonbon, on obtient précisément les coordonnées du coin supérieur gauche de la région (le point rouge). Vous pouvez vérifier : cette propriété est vraie dans n'importe quel cas de figure.

Sur la figure 2(b), les max sont obtenus en `col1` et `lig0`, autrement dit sur la colonne du bonbon orange et la ligne du bonbon vert. Cela correspond donc au bonbon qui serait situé entre le point bleu et le point vert, donc le bonbon le plus bas à droite dans la région englobante. Si l'on utilise les fonctions `getXPixel` et `getYPixel` sur `col1` et `lig0` respectivement, on obtiendra les coordonnées en pixels du point bleu, c'est-à-dire du coin supérieur gauche du bonbon. Ici, il vous faudra calculer les coordonnées du coin inférieur droit de ce bonbon (le point vert). Vous pourrez le faire aisément en utilisant le point bleu et la taille des images. Enfin, la différence de hauteur et de largeur entre le point vert et le point rouge vous donneront la hauteur en pixels `h` et la largeur `w` de la région.