Projet Tetris

0 - Introduction

Le but de cette activité de projet est double : il nous faut d'une part mettre en pratique les concepts du paradigme de programmation objet au sein d'une application (elle est codée en Java). D'autre part, comme cette activité se déroule en binôme, il faut mettre en place un système de collaboration, et la solution vers laquelle nous nous sommes tournés est GitHub. Cette plateforme nous permet de communiquer et partager facilement du code, ainsi que de le tenir à jour (grâce au logiciel de versionnement git).

1 - Piece.java par AFFANI Mehdi

L'un des premiers soucis auquel j'ai eu à faire avec piece.java est l'ArrayList Skirt. Parmi les premières fonctions à coder, je pense que c'est la plus compliquée. J'ai d'abord pensé à créer un tableau temporaire stockant les abscisses déjà rencontrées par la boucle for, puis après un moment j'ai réalisé qu'il manquait une condition importante que j'ai rajouté via un « else if ». Un peu plus tard, j'ai rencontré quelques autres soucis avec cette partie, après une rotation, puisque les coordonnées des points du Tetris n'étaient pas rangées dans le bon ordre. Puis j'ai eu du mal à visualiser l'effet d'une rotation sur une pièce, j'ai dû m'y prendre à plusieurs reprises avant de réussir. Les instructions du sujet me proposaient de faire une symétrie horizontale et verticale, ce que j'avais commencé par faire, mais je me suis rendu compte que ce n'était comme ça qu'une pièce agissait lors d'une rotation. En effet, après inversion des coordonnées des abscisses et des ordonnées il n'est pas nécessaire d'effectuer deux symétries, mais qu'une seule. Après symétrie par rapport à l'axe des ordonnées, la pièce a effectué une rotation contre le sens des aiguilles d'une montre.
Après avoir réussi, j'ai rencontré des soucis de dépassement des ArrayList à cause de la rotation. Les pièces ne s'alignaient pas à gauche après la rotation. Après l'avoir compris, j'ai pu modifier légèrement le code afin qu'il fasse en sorte que les pièces soient toujours alignées. J'ai ensuite eu des soucis avec les tests qui échouaient malgré le fait que les deux « body » comportaient les mêmes nombres. C'était parce que l'ArrayList Skirt était vide. Après avoir corrigé le problème avec Skirt, j'ai de nouveau eu droit à une erreur de dépassement. M'étant cassé la tête avec la rotation, et la fonction étant très peu lisible, j'ai décidé de la supprimer et recommencer de nouveau. Cette fois ci, j'ai décidé d'utiliser un tableau afin de pouvoir jongler avec les coordonnées plus facilement. Je comptais transformer le tableau en chaine de caractère que je passerai ensuite en paramètre d'une pièce. Bien sûr, ça ne s'est pas passé comme prévu, même si je m'attendais à une erreur puisque cette fois ci je n'avais pas mis en place de système d'alignement des pièces. Je me suis rendu compte que le système d'alignement n'était pas nécessaire, et que le problème venait d'ailleurs. Ce n'est plus une surprise, il y avait dépassement au niveau de Skirt. Au lieu de recommencer Skirt à zéro une autre fois, après avoir compris d'où venait le souci, j'ai modifié ma fonction

de rotation afin qu'elle trie les coordonnées par ordre croissant des abscisses des points qu'elle stockait. Restocker ces points dans un tableau qui serait ensuite converti en chaine de caractères me semblait un peu stupide, j'ai donc décidé de créer une ArrayList où y stocker les points rangés dans l'ordre croissant. Les autres fonctions n'ont pas posé beaucoup de difficultés, même si ParsePoints et toString ont nécessité beaucoup de recherches afin de mettre en œuvre les méthodes imaginées.

1b - Piece.java par FARES Jean-Marc

Il s'est avéré que deux versions ont été développées, et que la version précédente avait quelque bugs, notamment dans la méthode `toString();`, nous avons donc utilisé la seconde version à partir du commit `ff4a60b` (le changelog complet y est disponible). Le problème de la précédente implémentation de `toString()` était la gestion des indices dans le tableau intermédiaire, dans la mesure ou les dimensions du tableau et des points n'étaient pas compatibles (quand on cherchait un point de coordonnées (i, j), il n'existait pas car c'était (j, i) qui était dans la liste). Un autre changement notable est appliqué à la méthode `parsePoints(String rep)`, on utilise un pattern matcher et une RegEx pour trouver tous les couples de points de façon efficace. Il est ainsi plus simple d'écrire le constructeur `Piece(String points)` puisqu'on appelle tout simplement le précédent constructeur avec `parsePoints(points)`.

2 - Board.java par FARES Jean-Marc

Tout d'abord, il nous faut écrire les constructeurs de la classe, et le seul qui demande un effort est le constructeur par copie : en effet, comme certains de nos attributs sont des tableaux 2D, il nous faut les copier explicitement. Une manière élégante de le faire s'appuie sur les `Stream` de Java 8 : on crée un flux correspondant aux lignes du tableau puis pour chaque ligne, on la clone, et on enclave le tout dans un nouveau tableau. On aurait tout à fait pu procéder de même avec une boucle `foreach`. La fonction suivante, `int place(Piece piece, int x, int y)` est très simple, puisqu'il s'agit d'une suite de vérifications puis du placement de la pièce:

- Le plateau est dans l'état `committed`, sinon on envoie une `RuntimeException`
- Que les coordonnées de la pièce ne dépassent pas les limites du plateau
- Pour chaque coordonnée de la pièce :
  - S'il y a deja une pièce a cet endroit, on revoit `PLACE_BAD`
  - Sinon on change l'état de la case
  - Si celle-ci provoque le remplissage d'une ligne, on renverra `PLACE_ROW_FILLED`, `PLACE_OK` sinon.

La méthode `clearRows()` est découpée en deux sous-méthodes pour plus de simplicité :

- `clearOne(int y)` qui supprime tous les éléments d'une ligne remplie
- `dropFromRow(int y)` qui fait descendre tous les éléments qui se trouvent au dessus d'une ligne donnée

Pour effacer toutes les lignes remplies de la grille, on décide de créer une pile d'indices à effacer en parcourant une seule fois les lignes de la grille (on ajoute l'indice à la pile si la ligne est pleine). Ensuite, tant qu'il y a des éléments, on pop le dernier et on appelle les deux sous méthodes citées au dessus pour cet indice. Précédemment, nous nous contentions de determiner les indices des lignes a vider puis nous les vidions. Cette méthode fonctionne s'il n'y a qu'une seule ligne a vider puisqu'il faudrait mettre a jour l'attribut `heights` des qu'une suppression est effectuée, cependant, les indices des autres lignes a vider ne seraient plus les bons. Nous aurions pu généraliser cette dernière méthode, cependant elle amènerait à l'introduction de beaucoup de variables intermédiaires ; nous avons donc préféré la première méthode décrite, plus élégante.

On implémente de plus la méthode `dropHeight(Piece piece, int x)`, qui est simplement une recherche de maximum : on initialise un accumulateur à 0 et l'on calcule la quantité sur laquelle effectuer le maximum (ici, c'est la différence entre le point le plus haut de la colonne et le point le plus bas de la pièce). On renvoie l'indice avec le plus grand delta.

Les méthodes `undo()` et `commit()` sont simplement des copies de tableaux 1D et 2D, on procède exactement comme dans le constructeur de copie (cf. le code source pour plus de détails).

3 - JBrainTetris.java par AFFANI Mehdi

La difficulté principale de cette classe était la documentation. Une grande partie de ce qu'il fallait faire était assez bien expliqué dans l'énoncé du projet, malheureusement, si on n'avait pas lu et compris toutes les classes du projet, et effectué quelques recherches sur les interfaces, cases et autres fonctions utilisées dans cette classe, on avait du mal à comprendre ce qu'il fallait y faire. Du à l'utilisation de plusieurs fonctions de même noms, il était parfois nécessaire de mentionner de quelles classes on appelait certaines fonctions, ce que je n'avais pas l'habitude de faire. CreateControlPanel, où la deuxième fonction de cette classe sur laquelle j'ai travaillé, m'a posé des soucis à cause des raisons que j'ai donné plus haut. Je ne connaissais pas assez les fonctions que je pouvais utiliser. J'ai du m'y prendre plusieurs fois, et la modifier à plusieurs reprises avant de la recommencer de zéro quand il a fallu ajouter l'adversaire. Le problème principal était le changeListener qui m'a nécessité beaucoup de temps pour comprendre comment je pouvais l'utiliser correctement. PickNextPiece et PickWorstPiece j'ai codé en parallèle. J'ai mis un moment avant de comprendre que je devais comprendre et utiliser le suffixe « super » pour réduire le nombre de ligne de la fonction PickNextPiece et la faire fonctionner correctement.

4 - Conclusion

Afin de jouer a Tetris, il ne nous reste plus qu'à appeler la méthode `main(String... args)` de notre classe `JTetris` ou bien l'une de ses filles, comme par exemple `JBrainTetris`.
On remarque que l'intelligence artificielle sait bien placer une pièce pour compléter les lignes, cependant elle ne sait pas détecter les trous en dessous de blocs (qui nécessiteraient des déplacements latéraux de dernière minute).