Rapport Tetris

Explication du programme

Dans un premier temps, on initialise les variables responsables de la taille de la fenêtre et celles qui en dépendent selon si on est sur Linux ou sur un autre système d'exploitation. On génère ensuite les pièces et les couleurs de ces pièces. On rentre ensuite dans la fonction **main()** qui est appelée. Celle-ci est chargée de l'affichage du menu, ainsi que de la gestion des touches pour le choix dans un premier temps de lancer la partie ou de retenter puis en fonction du choix des variations et des bonus. En fonction, elle appelle la fonction game ou **gameMode2Joueurs** qui va lancer la partie.

Ces fonctions vont gérer la partie selon si vous êtes en mode deux joueurs ou non et selon si vous chargez une partie qui a été enregistrée ou non.
Dans un premier temps, la fonction game génère la **grille**, la **couleur** et les **pièces** ou charge une partie sauvegardée. Une boucle while représente le jeu et tourne jusqu'à ce qu'on quitte la partie. On appelle la fonction de mise à jour de l'affichage et ensuite celle de suppression des lignes. Si on est au début du jeu, on génère la pièce actuelle et la pièce suivante. Sinon, on passe la pièce suivante en nouvelle pièce et on génère la suivante et on appelle également la fonction **suppColor** dans le cas où le bonus de suppression par couleur est activé. Cette fonction est chargée de détecter les contacts de la pièce actuelle et s'ils sont avec une pièce de la même couleur et supérieurs à 2, on supprime alors les deux pièces. On gère ensuite la collision de la pièce grâce à la fonction **isColision** qui vérifie si la pièce active se superpose à une autre pièce, dans quel cas il y a **défaite** et donc la partie s'arrête.

Ensuite on appelle les fonctions correspondant aux bonus tels que **l'IA** ou encore le **mode pourrissement**. Le bonus IA va donc chercher le meilleur endroit pour poser les deux pièces suivantes, puis dans tous les cas on génère la pièce suivante aux coordonnées de départ grâce à la fonction **spawnPiece**. Après cela, dans le cas où on a activé la variable pourrissement, on va supprimer une pièce et réafficher la grille. On gère ensuite le temps pour la **variable de difficulté**. Dans ce if on s'occupe entre autres du temps qui s'écoule avant qu'on désactive une pièce, puis on redessine la grille. Enfin, si la variable de changement est à 1, c'est-à-dire qu'il y a eu un changement et qu'il faut redessiner la grille, on appelle la fonction **drawGrid** qui s'en charge.

Finalement, on gère les touches. Si on utilise la croix ou alt+f4 pour fermer, on sauvegarde automatiquement la partie et on passe le **flag** à quitter pour ne pas redessiner le **menu**. Sinon, si la touche est utile pour le jeu

et qu’on ne se trouve pas dans le bonus IA, on appelle alors la fonction **keyPressed()** qui gère les touches. Si c’est par contre la touche échap, on affiche le menu de pause, puis si on veut reprendre la partie, on affiche la grille grâce à **DrawGrid**, sinon on crée une sauvegarde et on quitte le jeu.

Finalement si nous avons perdu, la fonction **game()** renvoie la fonction **endScreen()** chargée d’afficher le menu de fin avec notre **score**.

Analyse de la complexité des algorithmes principaux

drawgrid() : pour une grille de i ligne et de j colonnes et qu’on suppose que les fonction de fltk sont en O(1) alors la fonction drawGrid est en O(i*j) grâce à l'attribution des couleurs pour chaque pièce qui est en O(1) puisque c’est juste une recherche d’indice.

suppcolor() : est en O(i*j*len(liste retournée)*SearchGrid + cases_accessibles + len(sameColor) + (pointsDiffColor ou pointsColor). On a dans un premier temps le parcours des cases de la pièce qui se fait en O(i*j*len(lst retournée)*SearchGrid) puis la recherche des cases_accessibles en O(cases_accessible) et le parcours du set qui nous est retourné pour effacer la cases en O(len(sameColor)) et enfin le calcul des points en O(pointsDiffColor) ou en O(pointsColor)

isCollision() est en O(i*j) pour une pièce de i lignes et j colonnes, car on compare si la valeur est à 0 en O(1) puis on vérifie si la case de la grille n’est pas vide en O(1) puis on fait un return en O(1).

keyPressed() est au pire des cas en O(i*j*drawPiece) car dans un cas on appelle rotatePiece qui appelle deux fois erasePiece qui est en O(i*j) puis drawGrid. Dans les autres cas on appelle simplement DrawGrid donc on est en O(drawGrid).

drawPiece() est en O(dimentions de la pièce) puisqu’on doit itérer dans la matrice qui représente la pièce.

spawnPiece est en O(drawPiece) car elle return drawGrid et c’est sa seule action.