Récursivité et filtre d'images

Fonction remplissage

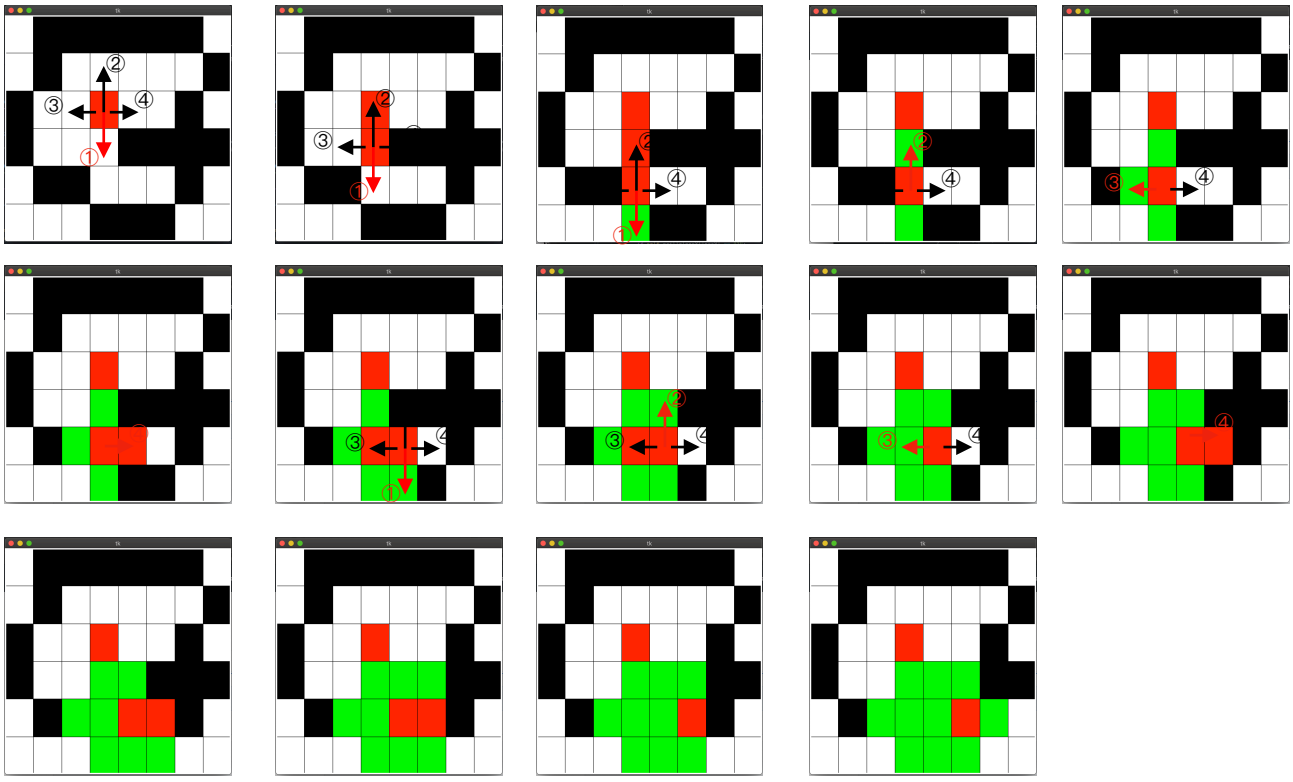
On ne se restreint plus à une image carrée. Partant d'un pixel donné par des coordonnées (par exemple (3,3)), la fonction de remplissage utilise un principe récursif pour remplir les zones adjacentes contenant des couleurs identiques.

Rappelons comment nous avions fait l'escalier de n marches en Turtle de manière récursive :
- on faisait une marche
- on se décalait
- on refaisait un escalier de n-1 marche
- quand on doit dessiner 0 marche, on s'arrête (condition d'arrêt)

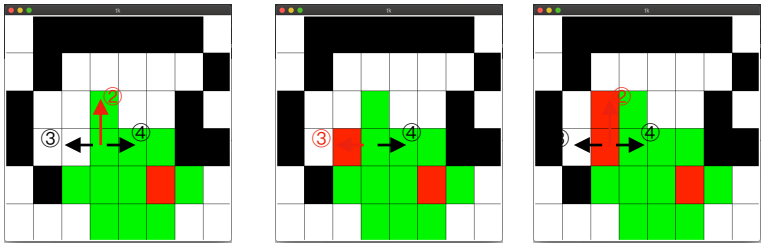
Nous allons appliquer le même principe en utilisant les valeurs des "pixels" pour définir les conditions d'arrêt. Pour colorier une zone blanche (couverte de 0 uniquement), nous allons procéder ainsi :
- nous colorions un pixel (nous mettons sa valeur à 2)
- nous colorions le reste de la zone blanche en commençant par regarder le pixel du bas, puis celui du haut, puis celui de gauche, puis celui de droite.
- quand on arrive sur un pixel dont la valeur est 1 **OU 2**, on s'arrête.

Rem : La condition pixel != 2 est essentielle pour éviter de recolorier un pixel déjà colorié. Dans ce cas, on rentrerait dans un appel récursif infini.

Voilà ce que donne les différentes étapes de l'algorithme :



On peut se dire qu'à la dernière étape, nous sommes bloqués. Sauf qu'il faut se souvenir de la pile d'exécution : nous remontons petit à petit dans la pile d'exécution jusqu'à trouver un cas qui ne correspond pas à une condition d'arrêt.



Au fur et à mesure, toute la zone va être coloriée mais cela va prendre beaucoup de temps!

Une proposition de code corrigé se trouve sur bouillotvincent.github.io . Je vous demande de bien étudier ce code et de comprendre comment fonctionne celui-ci. Essayez de refaire un exemple sur un autre cas.

Pour la taille de la pile, le plus simple est de créer un nouveau paramètre passé à la fonction. À chaque passage dans la zone d'appel récursif (else), on fait taillePile = taillePile + 1. Dans le cas où on passe par la condition d'arrêt, on enlève un niveau d'appel récursif (taillePile = taillePile - 1).

De cette manière, on peut voir qu'au maximum, la pile compte 11 fonctions en attente de condition d'arrêt !

Une autre méthode pour faire la même opération consiste à utiliser une **Pile** (comme une pile d'assiette). On prend la première case coloriée et on identifie tous ses voisins coloriables (pas des murs). On les enregistre dans un tableau.

pileDeCaseATraiter = [voisinBas, voisinHaut, voisinGauche, voisinDroit]

Tant que la pile de cases à traiter n'est pas vide, on choisit l'objet sur le sommet de la Pile : ici, le voisinDroit.

On colorie ce voisinDroit et on regarde si les voisins du voisinsDroit sont coloriables (pas des murs ou des cases déjà coloriées). Et on les ajoute au tableau de cases à traiter.

Ces structures (Piles, Files et Listes) vont être au coeur du projet chapitre que nous commencerons la semaine prochaine !