Licence 2 mono Informatique de Sorbonne Université

LU2IN006 Structures de données

RAPPORT PROJET
FloodIt

KERDOUCHE Nabil BOUYOUCEF Assirem

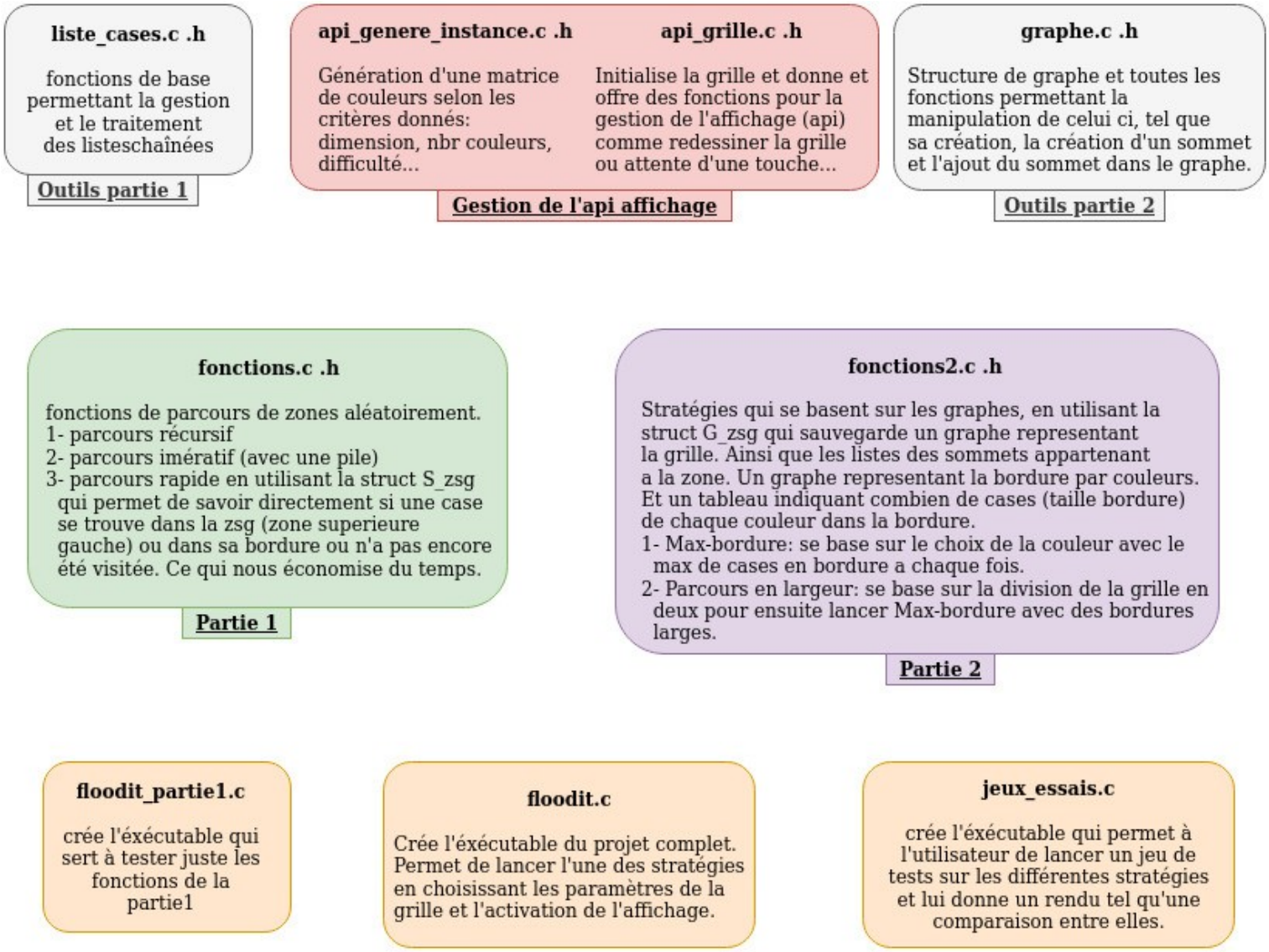
Groupe 1

18/05/2020

Introduction :

FloodIt est un jeu d'inondation de grilles de couleurs. Ce projet consiste à développer des stratégies pour gagner au jeu. Nous allons développer deux types de stratégies : des séquences aléatoires (partie1) et des séquences intelligentes pour faire le minimum de coups(partie2). Nous allons donc étudier chacune d'elles et les comparer sur plusieurs grilles différentes.

Schéma du projet complet :



NB :

-Les rôles de toutes les structures et fonctions sont décrits dans les fichiers .h

-Lors des tests de performance, désactivez l'affichage pour éviter un ralentissement

Pour l'exécution de floodit:

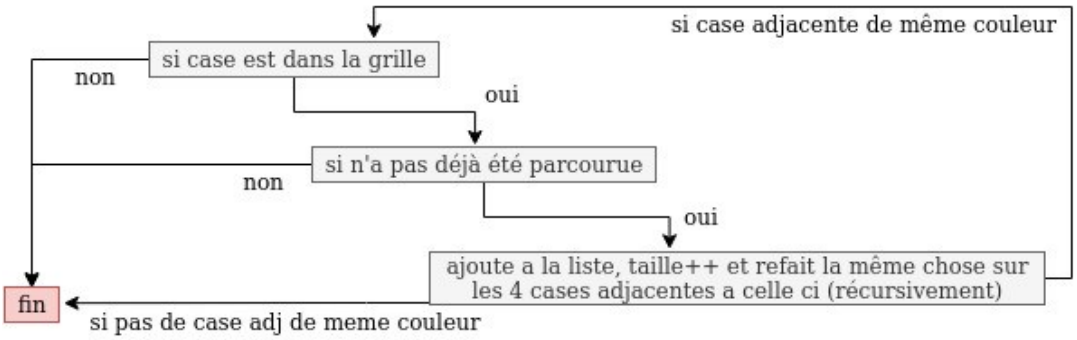
```
./floodit  dim  nbCouleurs  difficulte  graine  Strategie:1|2|3|4|5|6  aff:0|1
```

I- Description des algorithmes

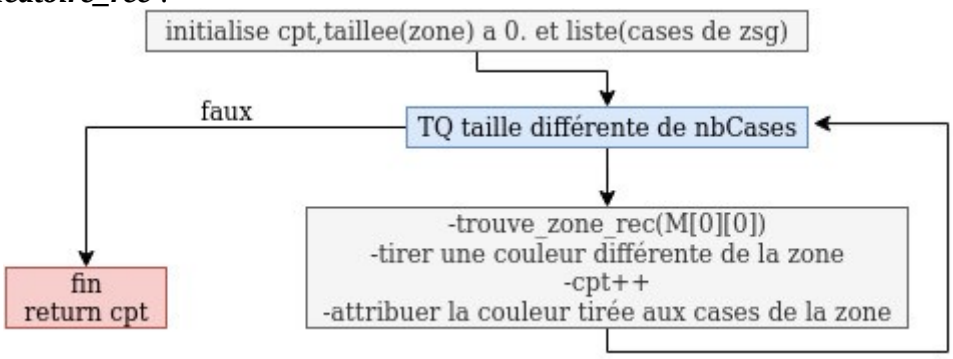
PARTIE 1 : Séquences aléatoires

1.1 : Aléatoire récursif:

trouve_zone_rec : à partir d'une case [i,j] trouve sa zone (cases de même couleur)

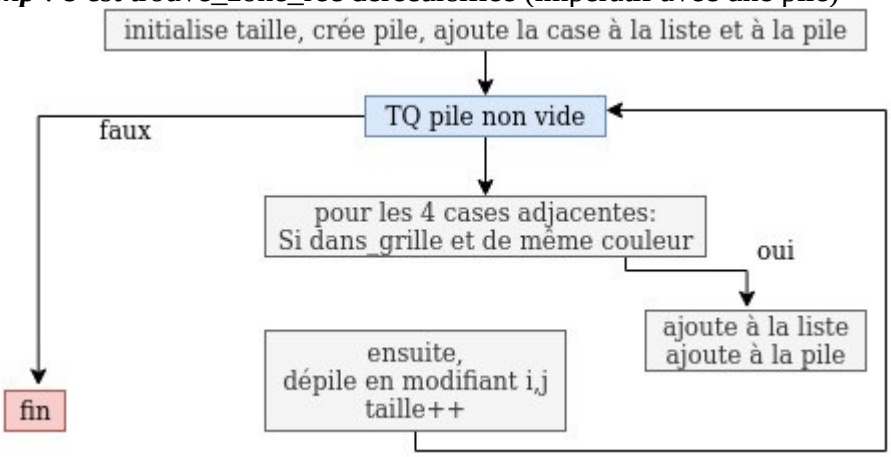


sequence_aleatoire_rec :



1.2 : Aléatoire impératif

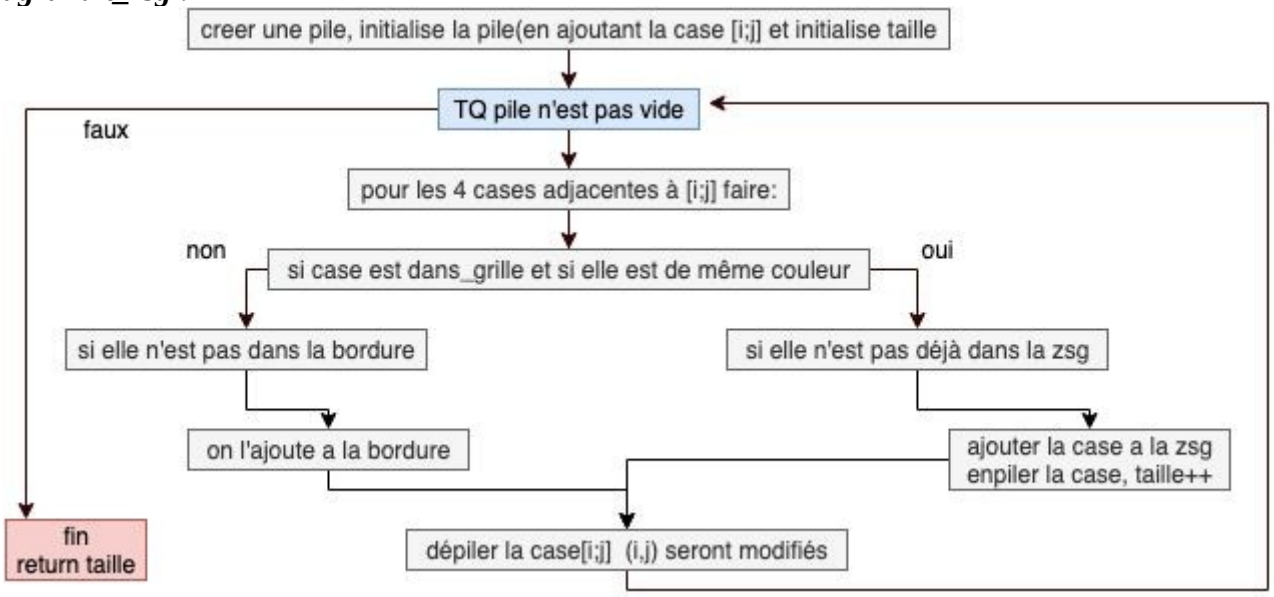
trouve_zone_imp : c'est trouve_zone_rec dérécursifiée (impératif avec une pile)



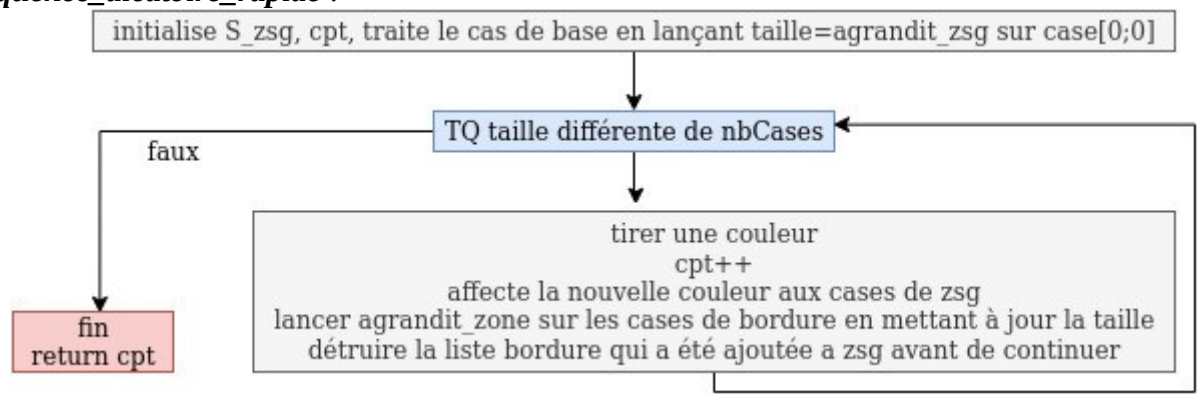
sequence_aleatoire_imp : exactement la même que sequence_aleatoire_rec sauf qu'elle utilise trouve_zone_imp (au lieu de rec) pour rechercher les zones.

1.3 : Séquence aléatoire rapide

agrandit_zsg :



sequence_aleatoire_rapide :



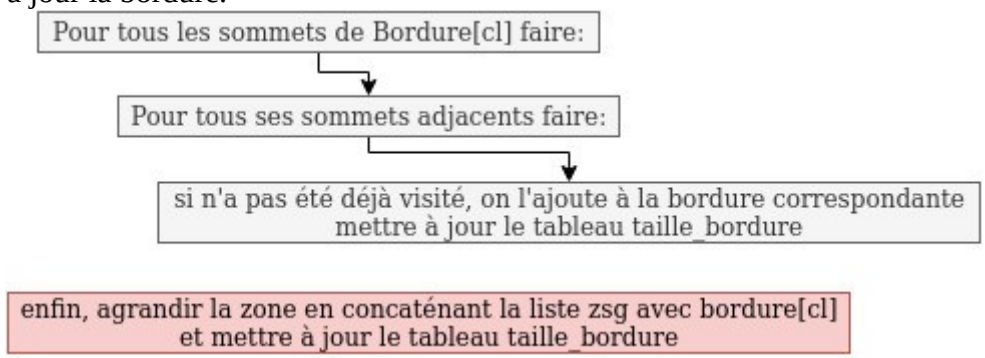
PARTIE 2 : Stratégies pilotées

2.1 : Graphe représentant la grille

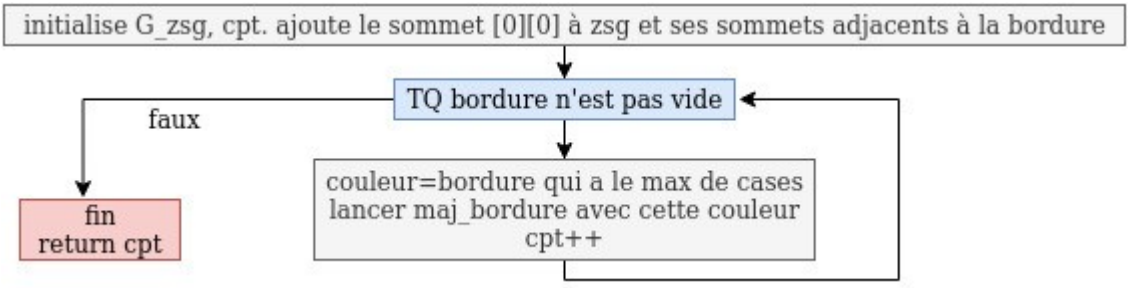
graphe.c .h contiennent la structure de graphe ainsi qu'un ensemble de fonctions permettant sa manipulation. Ce graphe sera utilisé pour les stratégies suivantes.

2.1 : Stratégie max-bordure – Description des algorithmes

maj_bordure_g : pour une couleur donnée, ajoute les sommets bordure de cette couleur dans la zone et met a jour la bordure.

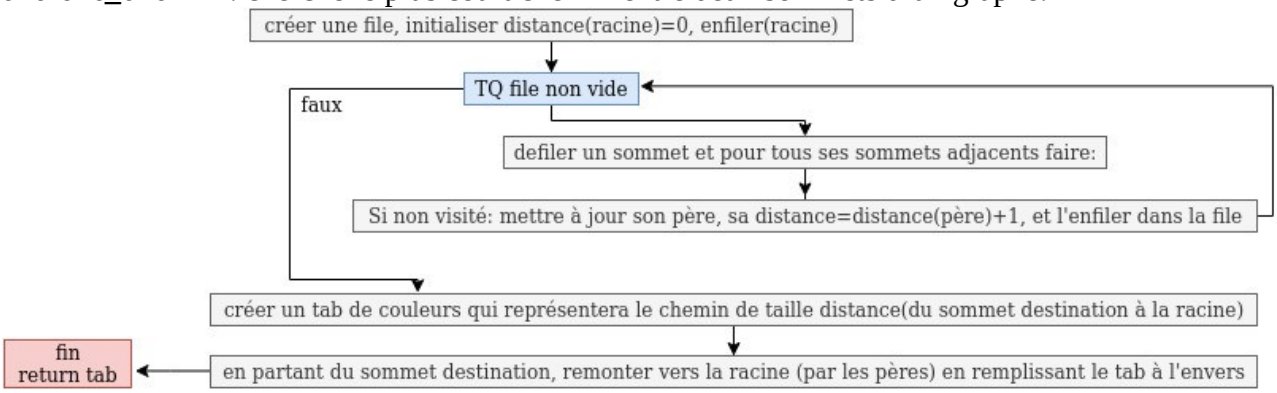


max_bordure :

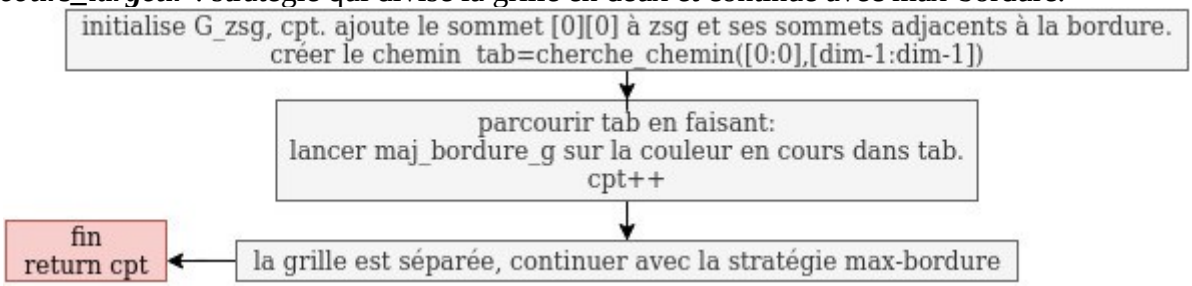


2.1 : Stratégie parcours en largeur – Description des algorithmes

cherche_chemin : cherche le plus court chemin entre deux sommets d'un graphe.



parcours_largeur : stratégie qui divise la grille en deux et continue avec max-bordure.

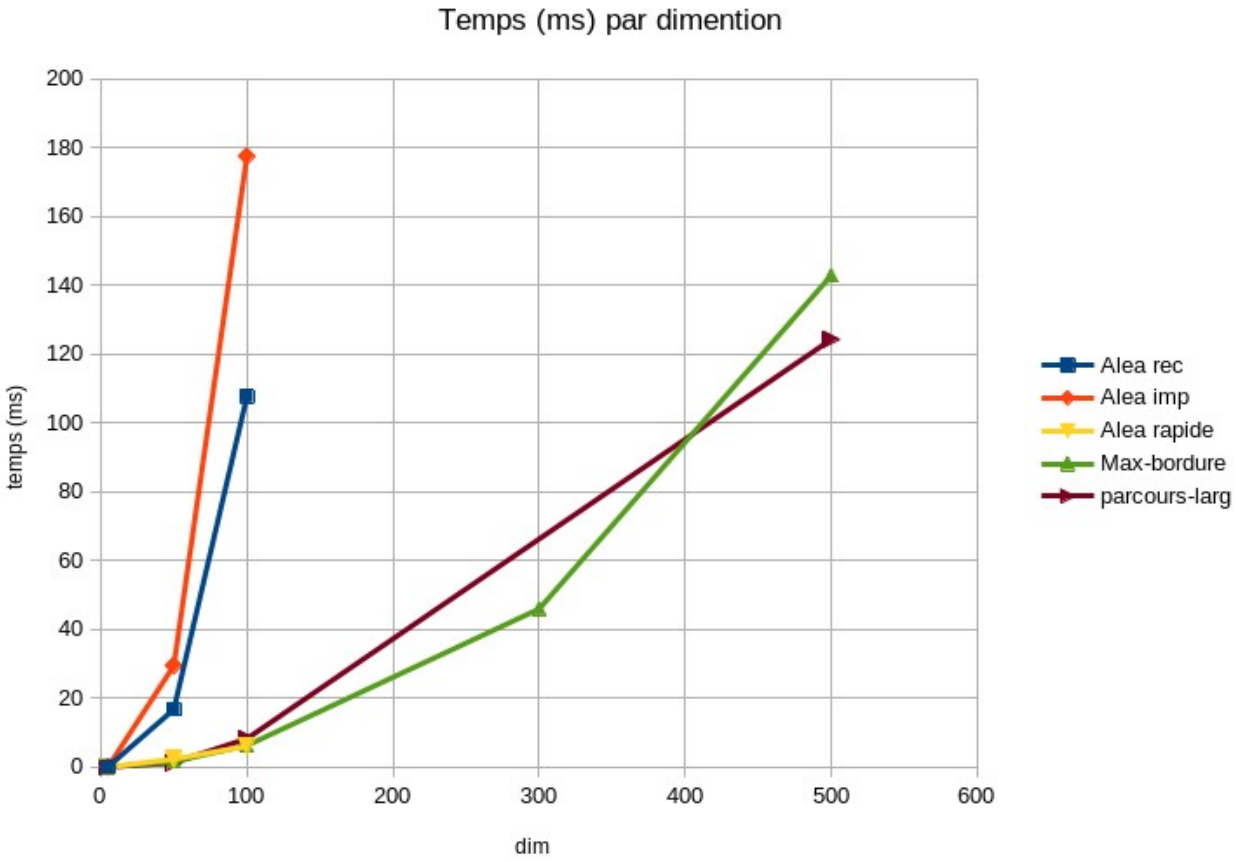


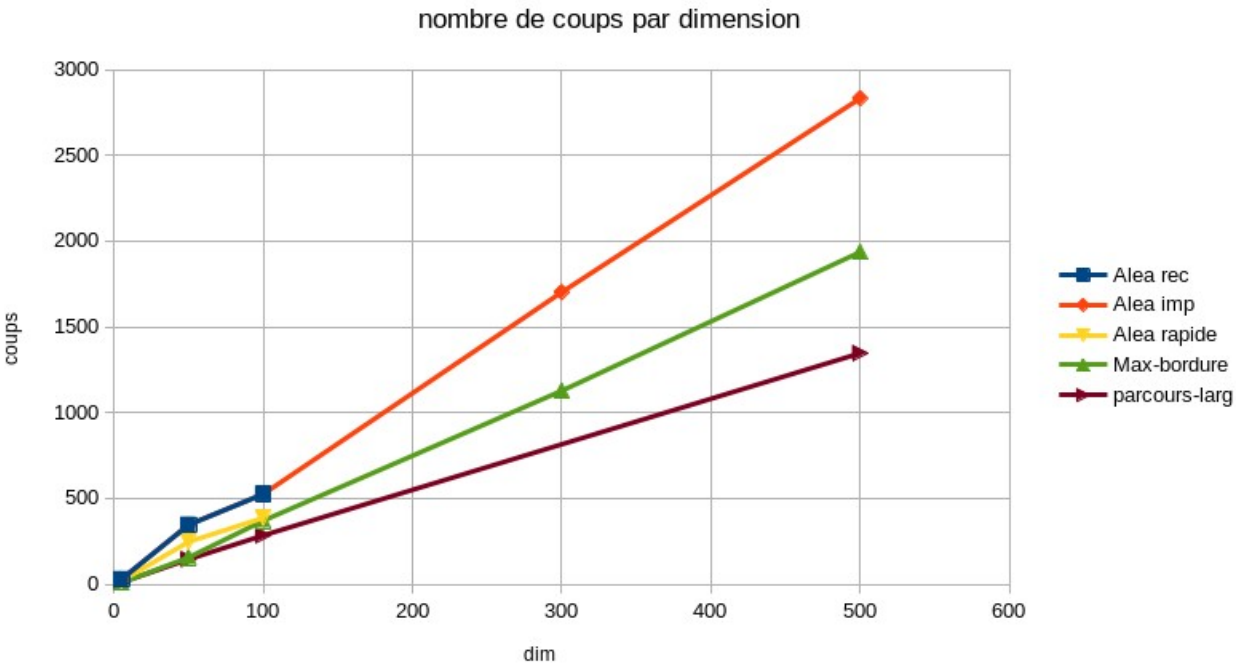
II- Statistiques

NB : format des cases des tableaux : Temps en **ms** (nombre de coups)

On varie la taille de la grille (dim)

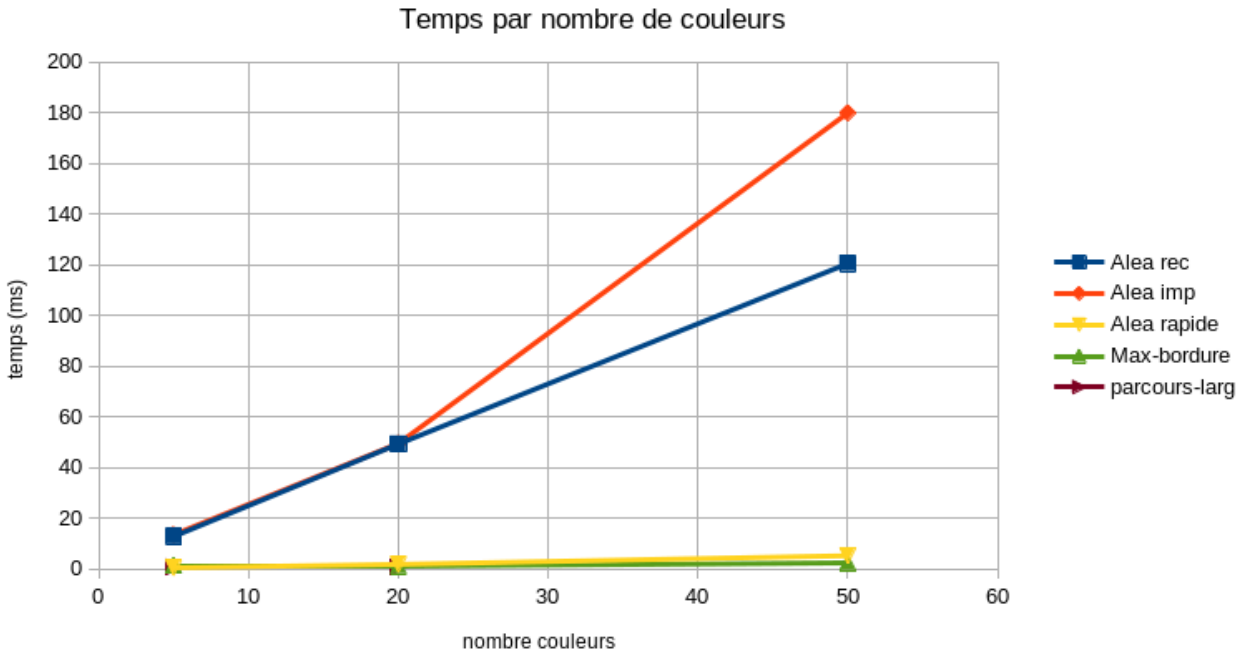
	Alea rec	Alea imp	Alea rapide	Max-bordure	parcours-larg
5 5 0 0	0,13 (29)	0,08 (29)	0,05 (19)	0,07 (8)	0,08 (10)
50 10 0 0	16,8 (346)	29,5 (346)	2,4 (247)	1,7 (156)	1,3 (145)
100 10 0 0	107,7 (525)	177,5 (525)	6,1 (386)	6,3 (370)	8,3 (284)
300 10 0 0	Segfault	9*e3(1702)	Segfault	46,5 (1127)	segfault
500 10 0 0	segfault	125*e3 (2832)	segfault	142,9 (1936)	124,3 (1346)

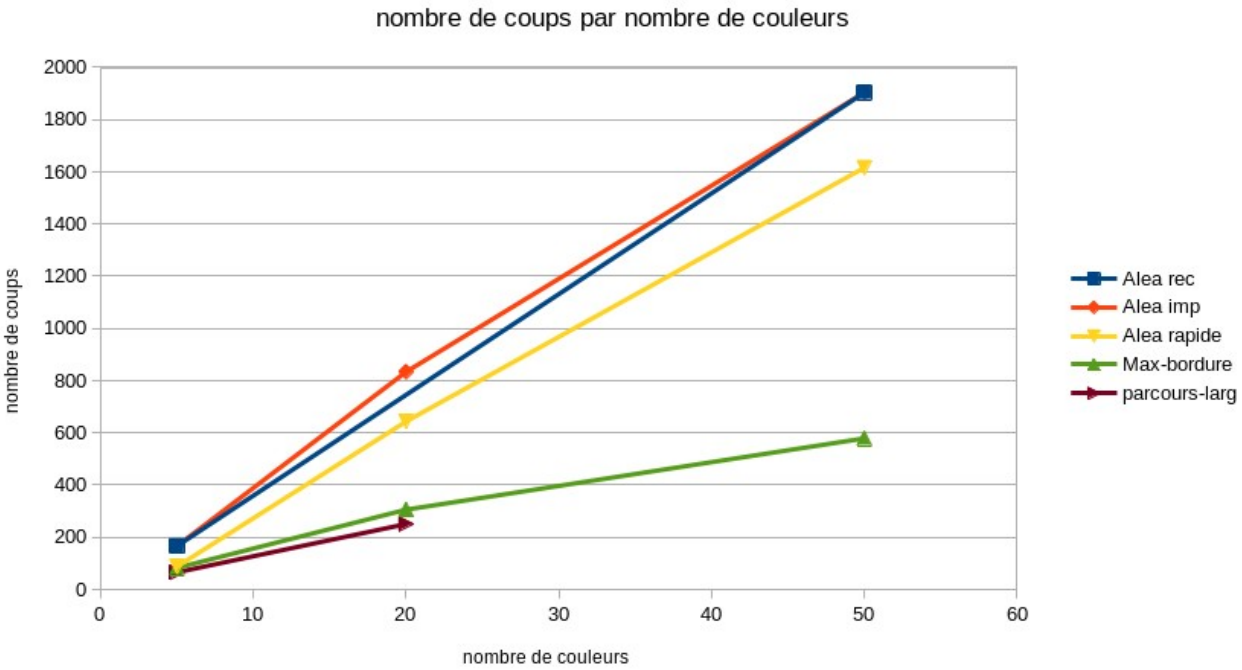

Temps (ms) par dimention
temps (ms)
dim
0
20
40
60
80
100
120
140
160
180
200
0
100
200
300
400
500
600
Alea rec
Alea imp
Alea rapide
Max-bordure
parcours-larg


nombre de coups par dimension
coups
dim
0
500
1000
1500
2000
2500
3000
0
100
200
300
400
500
600
Alea rec
Alea imp
Alea rapide
Max-bordure
parcours-larg

On varie le nombre de couleurs

	Alea rec	Alea imp	Alea rapide	Max-bordure	parcours-larg
50 5 0 0	12,9 (167)	13,6 (167)	0,58 (88)	1,5 (83)	0,9 (66)
50 20 0 0	49,4 (834)	49,6 (834)	1,9 (644)	1,2 (306)	0,9 (251)
50 50 0 0	120,6 (1904)	180,0 (1904)	5,3 (1616)	2,5 (579)	segfault





On varie la difficulté (diminue la difficulté)

	Alea rec	Alea imp	Alea rapide	Max-bordure	parcours-larg
50 20 0 0	49,4 (834)	49,6 (834)	1,9 (644)	1,2 (306)	0,9 (251)
50 20 5 0	24,8 (444)	27,7 (444)	1,7 (272)	1,1 (146)	1,2 (121)
50 20 15 0	18,3 (202)	27,7 (202)	1,4 (116)	0,5 (52)	0,4 (49)

Remarque : C'est tout à fait logique que toutes les fonctions se comportent de la même façon par rapport à la difficulté : elles sont plus efficaces quand la difficulté diminue, donc elles font moins de temps et moins de coups quand la grille est moins difficile.

III- Analyse

Aléatoire récursive: Cette version permet de résoudre le jeu mais la pile d'exécution risque de saturer avec une grille trop grande a cause des appels récursifs. En terme de vitesse elle est très lente, cela est due au tirage de couleurs aléatoirement qui augmente aussi le nombre de coups donc elle ne cherche pas a inonder la grille en un minimum de coups.

Aléatoire impérative : Elle est identique a la version Aléatoire récursive sauf qu'elle est dérécursifiée. Ce qui nous permet d'éviter la saturation de la pile d'exécution. Elles sont donc identiques en nombres de coups.

Aléatoire rapide : celle-ci se base sur une structure de données (S_Zsg) qui facilite l'identification des cases de la grille selon leurs appartenances a la zone, la bordure de la zone ou simplement a la grille en O(1) ce qui accélère le temps d'exécution mais pas le nombre de coups qui reste relativement proche au deux algorithmes précédents.

Max bordure : C'est un algorithme intelligent qui utilise la structure G_Zsg qui utilise un graphe, cela permet de répertorier la grille sous forme de zones contenant des cases adjacentes de même couleur, il ajoute ensuite a la zone la couleur dominante (plus grand nombre de cases de meme couleur) de la bordure. Cela permet d'inonder la grille en un nombre de coups plus ou moins optimal.

Parcours largeur : cet algorithme utilise la meme structure que max_bordure , au debut la recherche du plus court chemin entre le sommet supérieur gauche et le sommet inferieur droit cela permet de diviser la grille en deux et ensuite lancer max_bordure, cette stratégie permet de gagner en nombre de coups mais pas forcement en vitesse mais elle est très coûteuse en mémoire.

IV- Comparaison :

TEMPS/dimension : pour les quatre stratégie plus la dimension est grande plus le temps de traitement augmente. Les deux stratégies intelligentes (Max bordure et Parcours largeur) sont plus rapides , les stratégies aléatoires deviennent obsolètes a partir d'une certaine dimension de grille (environ 100*100) car la pile de alea_rec sature et le temps d'exécution d'alea_imp devient extrêmement lent. Concernant alea_rapide , elle se situe au milieu des deux types de stratégies.

COUPS /dim : Parcours_largeur inondes la grille avec un nombre de coups minimal. Max_bordure est moins efficace ensuite aléa_imp l'est encore moins. Aléa_rec offre le même nombre de coups que aléa_imp sauf qu'elle est inutilisable pour de grande grille.
TEMPS/nombre de couleurs : Les stratégies aléatoires sont très affectées par un grand nombre de couleurs. Cela est due au nombre d'essais qu'elles effectuent avant de trouver des cases adjacentes de la bonne couleur contrairement aux stratégies intelligentes qui sont très peu affectées.

COUPS/nombre de couleurs : Les stratégies aléatoires sont inefficaces pour optimiser le nombre de coups. C'est justement le point fort des stratégies intelligentes.

Conclusion:

Avant toutes chose, si l'on veut obtenir le minimum de nombre de coups, on oublie les séquences aléatoires. Pour des grilles de très petites tailles avec un nombre de couleurs peu élevé, il est préférable d'utiliser les stratégies aléatoires pour économiser de la mémoire vue qu'il n'y a pas de différence considérable de temps d'exécution. Pour des grilles larges et/ou ayant un grand nombre de couleurs, il est évident d'opter pour les stratégies intelligentes.