Rapport Color Flood

Lot-D

The A-Team

Loubna Abdellah

Anthea Bader

Projet IPI2

ENSIIE

Télécom Physique

Sommaire

I- Répartition du travail

1-L’équipe du travail

L’équipe du projet est constituée de 4 membres :

- Loubna Hennach (INOC)
- Anthea Mérida Montes de Oca (ENSIIE)
- Abdellah Elazzam (INOC)

- Bader Khatofi (INOC)

2-Le rôle de chacun des membres

Bader Khatofi:

- référent pour le lot D du projet
- gère la communication entre les membres de l'équipe
- organise les rencontres dans différents lieux (BNU, Biliothèque TPS …..)
- Chargé du diagramme de Gantt avec GanttPoject.

Loubna Hennach :

- Chargée de l'optimisation du solveur
- Chargée de la rédaction du rapport
- Chargée de la programmation de l'application complète en mode console

Anthea Mérida :

- Chargée de l'optimisation su solveur
- Chargée des tests unitaires
- Chargée de la programmation de l'application complète en mode console

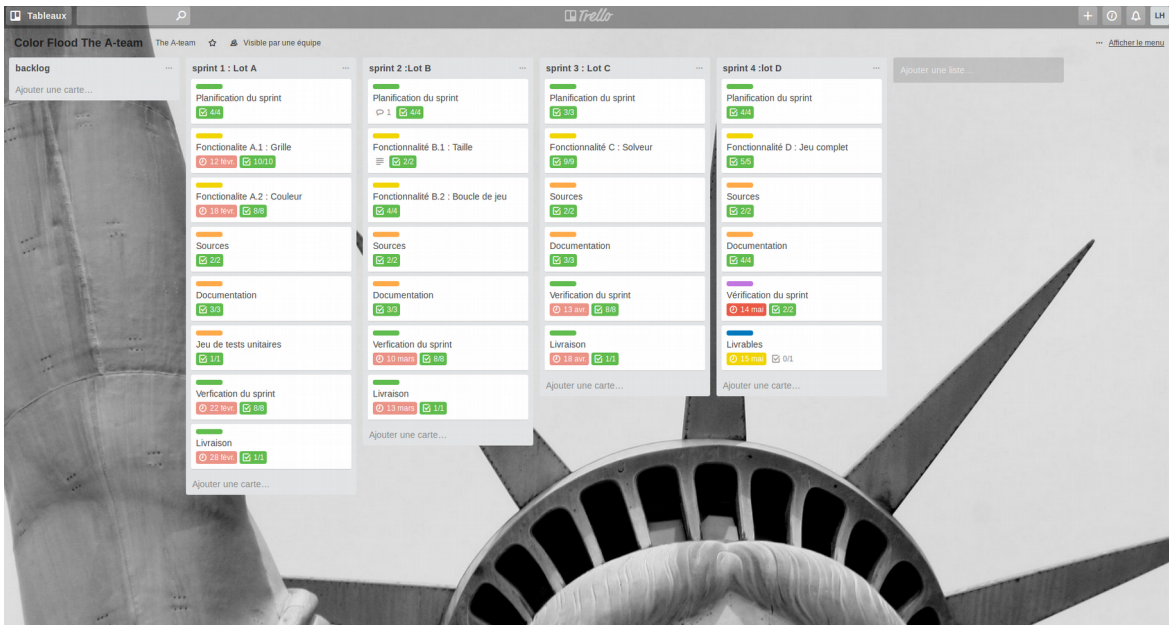
Abdellah Elazzam:

- Chargé de l’adaptation du solveur en mode graphique avec SDL
- Chargé de la programmation de l’application complète en mode graphique

4)La communication entre les membres

Nous nous sommes servis des outils qui avaient été mis en place pour les sprints précédents pour assurer la communication entre les membres de l’équipe.

Nous avons pallier les difficultés rencontrées au sprints précédents en nous réunissant fréquemment.



II – Conception algorithmique

1)Autre approche du solveur:

Le solveur conçu dans le lot C étant relativement lent, ainsi que les fuites mémoires qu'il générait à cause de la structure de piles de piles choisie, nous avons choisi d'adopter une nouvelle approche plus optimale. Cette approche consiste en la conception d'une fonction qui permet de sélectionner non seulement un choix pertinent c'est à dire une couleur adjacente à la composante connexe, mais aussi la couleur qui permet la plus grande expansion de la composante connexe ou la tâche.

La fonction étant heuristique elle permet donc une bonne résolution en temps ainsi qu'en nombre de coups.

Fonctions de base du solveur_rapide:

- grid copy(grid g) : permet de copier une grille passée en paramètre dans une autre grille g1, cette fonction sera très utile dans le déroulement de la solution et ce parce qu'on copie à chaque tour de jeu la grille qu'on vient de modifier dans une autre grille sur laquelle on effectue les changements : détection de la composante connexe, changement de couleur, rafraîchissement de la grille.

- bool choixpertinent(grid g,char c) : permet de savoir si une couleur parmi les six couleurs de l'application est adjacente ou non à la composante connexe (la tâche), elle nous permet donc de n'avoir à choisir que parmi les couleurs adjacentes et non les six couleurs.

- int compteur(grid g) : Ce compteur est mis en place pour calculer la taille de la composante connexe et ainsi il nous permet par la suite de comparer les couleurs qui permettent une plus grande composante connexe.

- char size_tache(grid g, char couleurs[6]) : permet de sélectionner la meilleur couleur adjacente à la composante connexe qui permet une plus grande tâche
  En effet, avec la fonction compteur on calcule la taille de la composante connexe (int tache=compteur(g)) de la grille passée en paramètre, on initialise une autre variable (int tache_bis=0) dans la quelle on stockera la taille de la composante connexe à chaque changement de couleur avec une couleur pertinente

```
change_color(&g2,couleurs[i]);
refresh_grid(&g2);
detect_flood(&g2,0,0,g2.array[0][0]);
```

  Ensuite on compare les tailles obtenues avec les couleurs pertinentes en sélectionnant à chaque fois celle qui permet la plus grande composante connexe

```
if(tache <= tache_bis)
{
        tache=tache_bis;
        couleur_res=couleurs[i];

}
```

  On veille à libérer la grille copie pour éviter les fuites de mémoire, et on retourne la couleur la plus pertinente.

- void solution_rapide(grid grille_depart,char sol[1000],int *i) : déroule récursivement la résolution d'une grille passée en paramètre grille_depart, on recopie la grille de départ dans une grille g2, on détecte la tâche puis on appelle la fonction size_tache qui nous retourne la meilleure couleur qu'on stocke dans un tableau de caractères passé en paramètres sol[1000] pour l'affichage de la solution, on change la composante connexe avec cette couleur et on appelle récursivement solution_rapide tant que la grille n'est pas résolue.
  On veille à libérer la grille copie pour éviter les fuites de mémoire.

2)L'application en mode console

L'application Color Flood est programmée en mode console ; Le joueur choisit la taille de la grille, il choisit ensuite le niveau de difficulté souhaité.
Le niveau de difficulté est défini de la manière suivante :
1/Niveau Facile : on rajoute 10 coups supplémentaires au nombre de coups générés par le solveur_rapide
2/Niveau Moyen : on rajoute 3 coups supplémentaires au nombre de coups générés par le solveur_rapide
3/Niveau difficile : on ne rajoute pas de coups supplémentaires au nombre de coups générés par le solveur rapide.

L'utilisateur est invité par la suite à saisir un caractère qui correspond à une couleur, ou 'Q' pour quitter la partie, ou 'H' pour avoir une solution de la grille en cours. Le nombre de coups autorisés diminue au fur et à mesure que le joueur avance dans le jeu.

Si le joueur résout la grille avec un nombre inférieur au nombre de coups autorisés il gagne la partie on lui affiche un message de victoire, sinon si il ne excède le nombre de coups autorisés on lui affiche un message d'échec , sinon dans le cas où l'utilisateur entre 'Q' il quitte la partie on lui affiche un message d'au revoir.

Fonctions de base de l'application:

- char get_colour() : permet à l'utilisateur de saisir une couleur. On demande à l'utilisateur de saisir un caractère tant que celui-ci ne correspond pas à une couleur, à 'Q' (pour quitter l'application), ou à 'H' pour afficher une solution calculée par le solveur.

- int get_size() : permet à l'utilisateur de saisir la taille de la grille désirée. On demande à l'utilisateur de saisir un entier tant que celui-ci n'est pas strictement supérieur à 1.

- int get_nombre_coups(grid g) : calcule le nombre de coups nécessaires pour aboutir à une bonne solution avec le solveur rapide.

- int get_level() : permet à l'utilisateur de choisir le niveau de difficulté et calcule le nombre de coups à rajouter en fonction de ce qui est saisi.

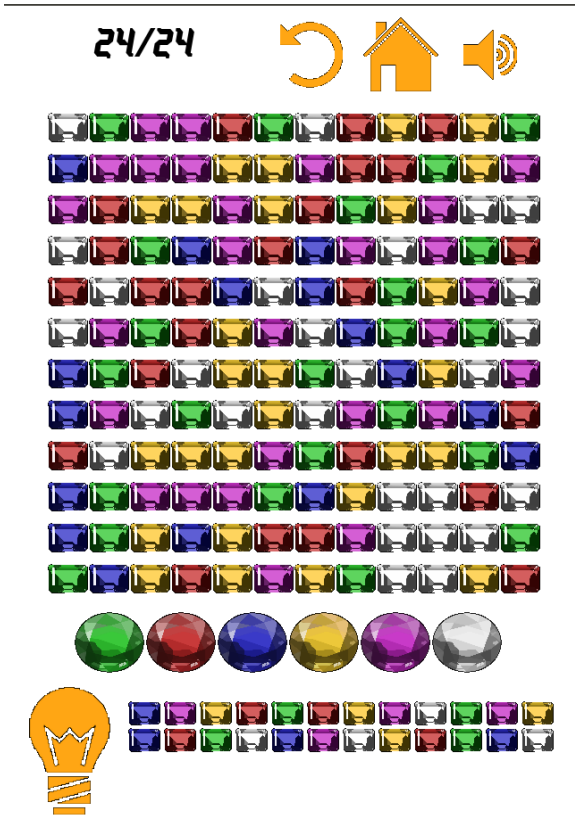
- void turn(int *coups_restants, int *nbr_mvm, grid *g, char c, int nombre_coups) : permet d'effectuer les changements majeurs à la grille à chaque tour de jeu.

- void check(grid *g, int test_quit, int nbr_mvm) : si toutes les cases de la grille sont de la même couleur et que l'utilisateur ne saisit pas « Q » on affiche un message de victoire. Si l'utilisateur dépasse le nombre de coups autorisés on sort de la boucle de jeu avec un message « Vous avez perdu ». Si l'utilisateur rentre « Q » on sort du jeu et donc partie non terminée

- void game() : dans une boucle while, on fait appel aux fonctions de base pour dérouler le jeu.

2)L'application en mode graphique

Difficultés :
→ L'affichage du jeu en mode graphique dépend de la résolution de l'écran utilisé. Chose qui a posé problème dans le présentation. Une capture écran du jeu en mode grahique a été fournie dans la présentation du projet pour une vue d'ensemble sur l'application.



→ Il y a des fuites de mémoires qui persistent malgré les libérations de mémoire dans plusieurs endroits tout au long du programme.

III. Documentation et tests mémoires

1)Vérification de l'absence de fuite mémoire:

On teste l'absence de fuite mémoire avec Valgrind pour l'exécutable jeu de l'application en mode console, cette dernière ne présente aucune fuite mémoire, le résultat renvoyé par valgrind est joint avec le code.

```
Suite: solveur
  Test: copy ... Taille non valide n <= 0
passed
  Test: compteur ...passed
  Test: choixpertinent ...passed
  Test: size_tache ...taille non valide
passed
  Test: solution_rapide ...taille non valide
passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      5      5      5      0        0
             asserts     29     29     29      0      n/a

Elapsed time =    0.039 seconds
```

2)Documentation du code:

Les commentaires en format doxygen sont inclus dans les fichiers.h (les prototypes), ceci nous permet de ne pas avoir de commentaires dupliqués quand on génère le rapport Doxygen.

Comme pour le lot C,le README contient toutes les instructions d'installation des outils mis en œuvre dans le lot D du projet.