Rapport projet IPI

Wassim Ben Youssef

Vendredi 14 Novembre 2014

Introduction :

Tout d'abord je dois préciser que pendant l'élaboration de mon projet, j'ai effectué de nombreux retour en arrière en me demandant si la solution que je proposait était optimale ou non, mais j'ai également été bloqué à de nombreuses reprises suites à mes choix qui s'avéraient ne pas être astucieux pour la suite. J'ai donc plusieurs fois été confronté à des erreurs de segmentation et à certaines absurdités qui s'affichaient à l'écran jusqu'à finalement obtenir un jeu qui compile et qui marche. Il n'est sûrement pas parfait et d'ailleurs je n'ai pas réussi à coder certaines fonctions, mais je pense que celles actuellement codées l'ont été avec le plus grand soin. A noter que j'ai séparé mon code en deux parties : le dossier pareil.c qui contient le main et le dossier pareil2.c qui contient les fonctions.

Choix de ne pas faire de struct :

Ma première idée était de faire un struct plateau (comme vu en TP) en mettant dans ce struct la hauteur et la largeur de plateau ainsi qu'un int** plat c'est à dire un tableau de tableau d'entier (ou une matrice d'entier de hauteur et de largeur définie). J'ai donc commencé par remplir mon plateau ainsi créée d'entiers aléatoires entre 0 et 3 puis j'ai créé une fonction qui transcrirai chaque entier en une des lettres A, O, H ou X. Mais cette solution est vite devenue lourde au fil des différentes fonctions que j'avais codé par la suite. J'ai donc changé ma matrice d'entier en matrice de char, toujours avec le même struct. Cependant, je me suis rendu compte qu'il n'était pas nécessaire de faire un struct et qu si je voulais créer une matrice remplie de lettres aléatoirement, il me suffisait de faire un char** dans la fonction main, d'initialiser une hauteur et une largeur que l'on utilisera dans chaque boucle for.

Remplir le plateau :

Pour remplir le plateau (ou la matrice) créée précédemment , il faut déjà commencer par allouer de la place pour chaque case. C'est ici qu'intervient la fonction 'remplissage' dans laquelle on crée une matrice de hauteur h et de longueur l et pour laquelle on alloue de la place pour contenir un char dans chaque case. On note alors que notre plateau est égal à cette matrice. Puis pour remplir cette matrice de lettre, on crée un tableau (appelé tableau dans la fonction main) de char composé des 4 lettres. Puis grâce à la fonction rand, on met dans chaque case une lettre tirée au hasard parmi ces quatre lettres. J'ai ici voulu créé une fonction qui donne un vrai aléatoire avec srand(time(NULL)) qui à l'origine donne un nombre totalement aléatoirement en basant ces calculs sur le nombre de secondes qui se sont écoulées depuis Janvier 1970 mais cette fonction ne donne un nombre totalement aléatoirement que toutes les secondes, or comme mon programme compile en moins de 1 seconde, on a alors la même lettre dans tout le plateau, c'est pourquoi j'ai gardé la fonction tableau[rand()%4] même si ce n'est pas un vrai aléatoire.

Fonction affichage :

Dans ma fonction affichage, j'ai essayé de mettre les coordonnées, mais après plusieurs tentatives je n'obtenait rien de bon (des chiffres s'affichaient un peu partout). J'avais donc essayé de créer une colonne et une ligne dans la matrice qui serai uniquement réservées aux coordonnées mais le problème de cette méthode est que étant donné que pl est une matrice de char, le décompte des coordonnées n'est effectué que jusqu'à 9. J'ai donc abandonné cette idée et je n'ai pas trouvé de solution pour afficher correctement les coordonnées. Quant à l'affichage de la matrice, j'ai fait simplement une boucle qui affiche les éléments de la matrice. J'ai également choisi de ne pas mettre les tirets dans la matrice mais de ne les afficher que après avoir afficher la matrice pour éviter des erreurs de calcul des coordonnées par la suite.

Boucle while :

J'ai codé une boucle while afin que le processus de jeu se répète jusqu'à ce que le jeu soit terminé. Cependant, dans mon cas, le jeu ne se termine que lorsque toutes les cases du plateau sont vide . Je n'ai pas réussi à traiter le cas où il n'y aurait plus de combinaisons possible dans le plateau. Pour terminer la boucle, j'ai donc mis une fonction fin que je détaillerait après qui ramène le score à 1000.

Les coordonnées :

Pour que l'origine des coordonnées soient située dans le coin en bas à gauche, lorsque l'utilisateur entre l'abscisse a et l'ordonnée o souhaitées, rien ne sera modifié pour l'abscisse mais pour l'ordonnée on aura h-o-1 dans le repère défini par défaut. Ainsi , si l'utilisateur veut sélectionner la case de coordonnées a et o dans son repère, on utilisera dans le code les coordonnées a et h-o-1.

La fonction dièse :

La fonction dièse permet de transformer tout les éléments communs à celui sélectionné en dièse. Pour cela, on commence par mettre la valeur de la case sélectionnée par l'utilisateur dans une variable tampon z. On compare ensuite chaque case voisine de la case sélectionnée : si la case d'à coté a la même valeur que la variable tampon z, cette case prend la valeur # puis on effectue le même test pour les voisins de cette case. On prend également soin de donner à la case sélectionnée la valeur #. Cependant, je n'ai pas réussi à coder une fonction qui redonnerait les valeurs initiales de chaque case si l'utilisateur décidait (dans la fonction supprimer qui suit) de ne pas supprimer de groupe sélectionné. De plus, cette fonction comprend certaines erreurs que je n'arrive pas à détecter : parfois aucune lettre ne devient #, d'autre fois des # apparaissent au mauvais endroit. Pourtant j'ai bien fait attention aux conditions : la fonction est bien récursive et pour chaque case on doit bien vérifier qu'elle appartienne bien au tableau.

La fonction supprimer :

La fonction supprimer est la fonction qui fait disparaître les dièses. Elle cherche simplement les dièses et les remplace par des espaces. Cependant j'étais obligé de mettre cette fonction dans le main car quand je la mettait dans le dossier pareil2.c des erreurs de segmentations apparaissaient. Il était donc plus logique pour moi de mettre cette fonction dans le main où ces erreurs ne se produisaient pas.

La fonction descendre :

La fonction descendre fait descendre les lettres après la disparition des dièses. Pour cela, je prend un pointeur sur chaque case de la matrice. Si on a un espace dans la case, le pointeur va pointer vers la case située au-dessus. On fait ensuite tourner cette fonction le nombre de fois la hauteur du plateau afin d'être assurée que toutes les cases sont bien descendu, cependant cette solution n'est sûrement pas très bonne car on fait tourner la fonction plus de fois que nécessaire.

La fonction fin :

La fonction fin permet de sortir de la boucle while. Elle n'est cependant pas fonctionnelle car elle ne marche que si j'avais réussi à coder la fonction qui permet de décaler les colonnes vers la gauche lorsqu'une colonne est vide. En effet, si cette fonction était codée, toutes les cases seraient vides si et seulement si la case tout en bas à gauche est vide et alors le score serait amené à 1000 ce qui termine le jeu. Je n'ai pas également réussi à coder le fait que le jeu est fini lorsqu'il n'y a plus de combinaisons possible.

Conclusion :

Suite à ce projet, je pense avoir assimilé certaines bases du langage C même si quelques difficultés persistent. Je tient également à préciser que j'ai fait plusieurs cessions de travail avec Ahmed Hermi et Tanguy Benkritly.