MA DOCUMENTATION PERSO POUR LNCURSES

ANNEXE DE MON DOSSIER DE CODAGE DU 2048 EN C++

Quelques bases

Au début de chaque fichier qui utilise ncurses, inclure la **librairie** :

```
#include <ncurses.h>
```

Le terminale peut être défini par 2 types de fenêtre, le `curscr` pour Current Screen, ou `stdscr` pour Standart Screen.
Pour afficher/actualiser notre fenêtre, on va utiliser `refresh()` pour `stdscr` ou `wrefresh()` pour un `curscr`. On initialise la fenêtre avec la commande `initscr()` .

Pour afficher un int ou un texte dans stdscr, on va utiliser addch :

```
Addch("Mon premier texte");
Addch(893);
```

La plupart des fonctions sont paramétrées par défaut pour le `stdscr`, comme `addch`. Sinon c'est `waddch`. On verra souvent un w devant le nom de la fonction dans ce cas.

La fenêtre est placé par rapport à un point en haut à gauche de coordonnées (y,x). Afin de décaler cette fenêtre, on va utiliser `move()` ou `wmove()`. On peut ensuite combiner les fonctions (où x et y deux nombres, ch une chaîne de caractère).

```
move(y,  x) ;
addch(ch) ;
```

Ou ensemble :

```
mvaddch(y, x, ch);
```

Avec une fenêtre spéciale dont le nom serait win :

```
mvwaddch(win, y, x, ch) ;
```

La commande `getch()` placé après un `refresh()` permet d'attendre que l'utilisateur appuie sur une touche avant de passer à la suite.

Ne pas oublier de terminer le programme par `endwin()` qui permet de tuer la fenêtre de ncurses afin tout bug du terminal. On le placera juste avant le return 0 par automatisme pour simplifier les choses.

Variables

LINES est un int qui contient le nombre de lignes du terminal
COLS de même pour le nombre de colonnes
On a toujours le type bool qui contient True et False, mais aussi **ERR** une erreur (-1) ou **OK** quand tout va bien.
strlen(string message) est un entier dont la valeur est la longueur de message (nombre de caractères). Utile pour centrer un texte dans la console.

Prendre en compte la touche pressée

On peut demander de déclarer un entier c qui prend une certaine valeur en fonction de la touche sur laquelle appuie l'utilisateur. Par exemple, si l'utilisateur appuies sur A, l'int c prendra la valeur 97, B donnera 98, … La touche pressée s'affichera ainsi que son numéro.
Voici un tableau de quelques touches et leurs valeurs :

Touche	Numéro	Touche	Numéro
A	97	**0**	48
B	98	**1**	49
E	101	**9**	57
R	114	**Flèche Bas**	258
Z	122	**Flèche Haut**	259
Espace	32	**Flèche Gauche**	260
Entrer	10	**Flèche Droite**	261

Toutefois, pour que les flèches du clavier soient bien considérées comme les 4 numéros ci-dessus, il faut impérativement activer le Keypad comme ceci :

```
keypad(stdscr, TRUE);
```

On glissera ce codee juste après l'ouvertue de notre fenêtre, au début.

Pour obtenir ce numéro, on utilisera la commande getch ainsi :

```
int c = getch();        //c prend la valeur de la touche entrée
printw("%d", c);  //%d signifie aller prendre la valeur de la var, soit c
```

Squelette de programme

Notre premier code, ne contenant aucune fonction annexe ou instruction, seulement le squelette, ressemblera à ceci :

```
#include <ncurses.h>          //obligatoire pour utilizer ncurses
using namespace std;

int main(){
  initscr();                  //pour ouvrir une fenêtre de ncurses
  keypad(stdscr, TRUE);       //active les flèches

      [INSTRUCTIONS]

      refresh();              //actualise la fenêtre (pas obligatoire ici)
      getch();                //attend  que  l'utilisateur  appuie  une  touche,
comme cin mais pour une seule touche

  endwin();                   // ferme la fenêtre ncurses, pour éviter bug
  return 0;
}
```

Premiers Pas

Voici un programme qui affiche la valeur d'une touche lorsqu'on appuie dessus (qui a servit à remplir le tableau ci-dessus). On notera que "`\n`" est équivalent à `endl`.

```
#include <ncurses.h>
using namespace std;


int main(){
initscr();
keypad(stdscr, TRUE);

      int c;

      printw("Appuyer sur une touche :\n");

      c = getch();      //va afficher la touche appuyée
      printw(" %d \n", c);    //affiche la valeur

      getch();

endwin();
return 0;
}
```

Effacer la fenêtre

Afin d'effacer toute la fenêtre, on utilisera la commande `clear()` . Ceci va réinitialiser la fenêtre (effacer et remettre le curseur au début).

Ecrire où on veut dans la fenêtre / Centrer du texte

Lorsque l'on écrit du texte dans la fenêtre de ncurses, il se met par défaut aux coordonnées (y, x)=(0, 0) , soit en haut à gauche. Or il est possible de décaler l'affichage. Si on veut par exemple commencer un texte à 10 caractères vers la droite, et 3 lignes vers le bas, on utilise la commande:

```
move(3, 10);
```

Cela évite de faire 3 endl puis 10 espaces. Faire attention, il faut bien donner les coordonnées sous le format **(y, x) et pas (x, y)** ! On peut également utiliser strlen pour centrer le texte. Si notre chaîne de caractère s'appelle *msg*, alors si celle-ci contient 25 caractères, on aura ce code pour l'afficher centré :

```
int centrage = strlen(msg) ; //centrage = 25
move(0, COLS/2 - centrage/2);
printw(msg);
```

On peut aussi contracter en une ligne :

```
int centrage = 25 ;     //si on sait à l'avance que c'est 25
mvprintw(0, (COLS / 2) - (centrage / 2), msg);
```

Quelques problèmes de compatibilités

Pour déclarer notre string, on doit modifier le type, car il n'est plus compatible avec ncurses. Afin qu'il soit accepté autant par le c++11 que par ncurses, on va utiliser

```
char const *ligneSeparation = "mon texte ici" ;
```

Au lieu de :

```
string ligneSeparation = "mon texte qui bug" ;
```

De plus, notre ligne de compilation en console devient :

```
g++ -std=c++11 nomDuFichier.cpp –o NomDeSortie -lncurses
```

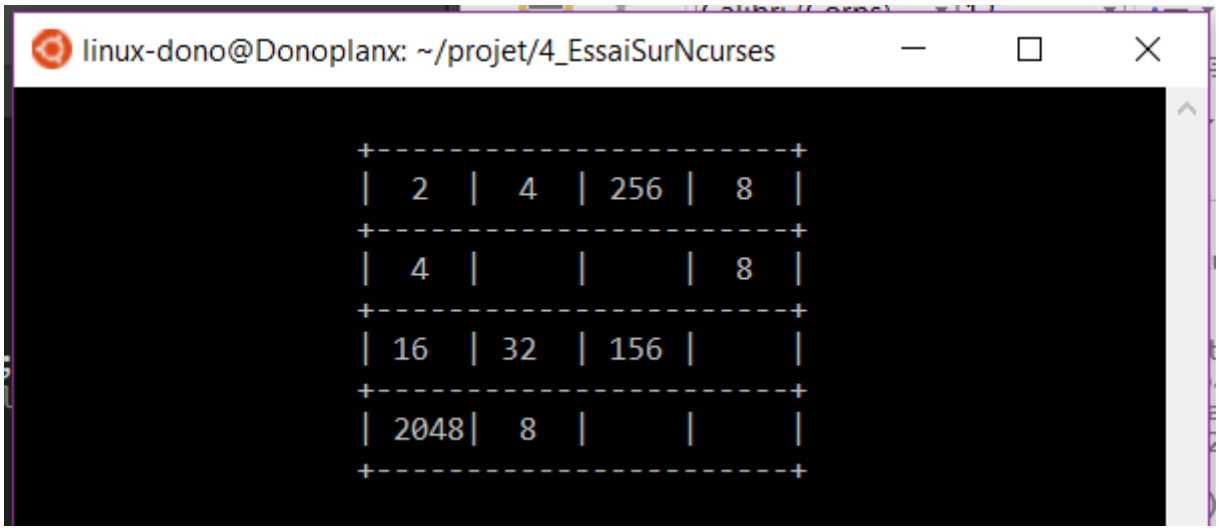
Affichage du plateau de jeu

Nous allons faire de nouvelles fonctions annexes pour afficher notre plateau de jeu 2048. Cette fois nous n'allons pas utiliser des étoiles * pour les bordures du tableau mais des " + - l ". Elle sera sujette à des changements par la suite lorsque l'on introduira les couleurs.

```
#include <ncurses.h>
#include <iostream>
#include <vector>
using namespace std;

void afficheCase(int valeurCase) {
	// Cette fonction est très manuelle, elle sera modifiée en fin de projet
afin
	// de l'automatiser au lieu d'y aller au cas par cas
	if( valeurCase == 0 ) {
		printw("     ");
	} else {
	if( valeurCase < 10 ) {
		printw("  %d  ", valeurCase);
	} else {
	if( valeurCase < 100 ){
		printw(" %d  ", valeurCase);
	} else {
	if( valeurCase < 1000 ) {
		printw(" %d ", valeurCase);
	} else {
		printw(" %d", valeurCase);
	}
	}
	}
	}
	}
}


void afficheLigne(vector<vector<int> > plateau, int ligne){
	printw("|");
	afficheCase( plateau[ligne][0] );
	printw("|");
	afficheCase( plateau[ligne][1] );
	printw("|");
	afficheCase( plateau[ligne][2]) ;
	printw("|");
	afficheCase( plateau[ligne][3] );
	printw("|\n"); //retour à la ligne


}
```

```
void affichePlateau(vector<vector<int> > plateau) {
      char   const   *ligneSeparation   =   "+----------------------+\n"   ;
//initialise la ligne de séparation
      mvprintw(1, (COLS/2) - (25/2), ligneSeparation) ; // Centre le plateau à
la deuxième ligne
      move(2, (COLS/2) - (25/2)) ;
      afficheLigne(plateau, 0);
      mvprintw(3, (COLS/2) - (25/2), ligneSeparation) ;
      move(4, (COLS/2) - (25/2)) ;
      afficheLigne(plateau, 1);
      mvprintw(5, (COLS/2) - (25/2), ligneSeparation) ;
      move(6, (COLS/2) - (25/2)) ;
      afficheLigne(plateau, 2);
      mvprintw(7, (COLS/2) - (25/2), ligneSeparation) ;
      move(8, (COLS/2) - (25/2)) ;
      afficheLigne(plateau, 3);
      mvprintw(9, (COLS/2) - (25/2), ligneSeparation) ;
      refresh();

move(11,0);
}
```

Je ne suis pas complètement satisfait de ces fonctions qui sont un peu trop manuelles et pas assez automatisée à mon goût … Mais je vais devoir les changer une fois arriver au chapitre des couleurs, donc je ne l'améliore pas encore.
Voici le résultat avec un exemple de plateau quelconque, centré sur la console :



Un peu de couleur

Avant de lancer les commandes, il faut s'assurer que notre terminale va supporter les couleurs. Pour cela, on va utiliser la commande has_colors() qui renvoie True si le terminale les supporte. On commencera ainsi par le code :

```
if (!has_colors() ) {
      printw("Erreur : Le terminal ne supporte pas les couleurs.");
      getch();
      return -1; //renvoie une erreur
}
```

Si ce test est passé, on active les couleurs avec la commande suivante :

```
start_color();
```

Ce qui nous intéresse sera de modifier un affichage de printw. On va pour cela utiliser 2 commandes, placées respectivement avant et après notre printw, afin d'activer puis désactiver les couleurs. Il s'agit de `attron(…)` et `attroff(…)` pour Attribut ON/OFF.
La valeur de ces attr sera de la forme `COLOR_PAIR(n)` où n sera un entier entre 1 et 999.
Chaque `COLOR_PAIR` contiendra 2 infos, le Foreground et Background (couleur de police et couleur d'arrière-plan). Ces valeurs sont par défauts Blanc et Noir (écrit en Blanc sur Noir).

Les valeurs possibles de couleurs sont : BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE. Cela fait 8 couleurs disponibles, On aurait aimé en avoir 10 pour tous les numéros de 2 à 2048. Mais pour l'instant, on se contentera des 8 couleurs, en mettant la même couleur pour toutes les hautes valeurs.

Bref, on veut changer la couleur !

On va procéder par étape :
1 : avec init_pair(**n**) , on va créer une pair de Foreground et Background
2 : on écrit notre attron, printw et attroff
3 : Les attributs prennent la valeur COLOR_PAIR(**n**)

Voici un exemple. On veut écrire en Rouge sur Blanc :

```
init_pair(1, COLOR_RED, COLOR_WHITE);
attron( COLOR_PAIR(1) );
printw("Le texte coloré rouge sur blanc");
attroff( COLOR_PAIR(1) );
```

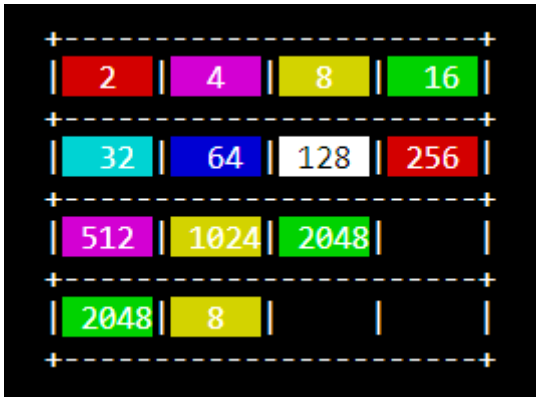
Je ne suis pas sûr que `init_pair` fonctionne dans une fonction annexe, donc par précaution, on va peut-être devoir les déplacer dans le main. Pour l'instant, on essaie de les mettre dans une fonction annexe à main.
On va donc modifier la fonction "afficheCase" présente un peu plus haut, afin d'attribuer à chaque valeur sa couleur.

Couleur dans le 2048

Dans notre 2048, j'ai donc refait la fonction, en y allant cas par cas pour affecter à chaque valeur sa couleur. Le code est long, je le mets donc en police plus petite ci-dessous :

```
void afficheCase(int valeurCase) {
	init_pair(1, COLOR_WHITE, COLOR_RED);
	init_pair(2, COLOR_WHITE, COLOR_MAGENTA);
	init_pair(3, COLOR_WHITE, COLOR_YELLOW);
	init_pair(4, COLOR_WHITE, COLOR_GREEN);
	init_pair(5, COLOR_WHITE, COLOR_CYAN);
	init_pair(6, COLOR_WHITE, COLOR_BLUE);
	init_pair(7, COLOR_BLACK, COLOR_WHITE);

	if( valeurCase == 0 ) {
		printw("     ");
	} else {
	if( valeurCase == 2 ) {                    // 2
		attron( COLOR_PAIR(1) );
		printw("  2  ");
		attroff( COLOR_PAIR(1) );
	} else {
	if( valeurCase == 4 ) {                    // 4
		attron( COLOR_PAIR(2) );
		printw("  4  ");
		attroff( COLOR_PAIR(2) );
	} else {
	if( valeurCase == 8 ) {                    // 8
		attron( COLOR_PAIR(3) );
		printw("  8  ");
		attroff( COLOR_PAIR(3) );
	} else {
	if( valeurCase == 16 ) {                   // 16
		attron( COLOR_PAIR(4) );
		printw("  16 ");
		attroff( COLOR_PAIR(4) );
	} else {
	if( valeurCase == 32 ) {                   // 32
		attron( COLOR_PAIR(5) );
		printw("  32 ");
		attroff( COLOR_PAIR(5) );
	} else {
	if( valeurCase == 64 ) {                   // 64
		attron( COLOR_PAIR(6) );
		printw("  64 ");
		attroff( COLOR_PAIR(6) );
	} else {
	if( valeurCase == 128 ) {                  // 128
		attron( COLOR_PAIR(7) );
		printw(" 128 ");
		attroff( COLOR_PAIR(7) );
	} else {
	if( valeurCase == 256 ) {                  // 256
		attron( COLOR_PAIR(1) );
		printw(" 256 ");
		attroff( COLOR_PAIR(1) );
	} else {
	if( valeurCase == 512 ) {                  // 512
		attron( COLOR_PAIR(2) );
		printw(" 512 ");
		attroff( COLOR_PAIR(2) );
	} else {
	if( valeurCase == 1024 ) {                 // 1024
		attron( COLOR_PAIR(3) );
		printw(" 1024");
		attroff( COLOR_PAIR(3) );
	} else {
	if( valeurCase == 2048 ) {                 // 2048
		attron( COLOR_PAIR(4) );
		printw(" 2048");
		attroff( COLOR_PAIR(4) );
	} else {
		printw("Erreur :Valeur de la case incorrect");
	} } } } } } } } } }  } } } // !! Cette ligne peut entraîner des erreurs
                                   // si il n'y a pas le bon nombre d'accolades
}
```

Voici un aperçu d'un plateau obtenu avec toutes les valeurs possibles :



Vers une nouvelle version

Pour pouvoir utiliser les nouvelles fonctions d'affichages (qui fonctionnent, testées sur un exemple ci-dessus), il faudra tout revoir. Chaque cout devra être convertit dans ncurses par des printw. Tout cela se fera dans un nouveau dossier afin de garder une version intact du jeu sans ncurses.
Quelques modifications sont aussi à apporter au fichier .h.

Pour éviter tout problème de compatibilité, comme vu plus haut, tous les String sont remplacés par des char const*.
Un des autres changements majeurs, est la valeur de la commande. Avant, on demandait à l'utilisateur d'entrer une chaîne de caractère. Maintenant, Lorsqu'il appuie sur une touche, la valeur de la touche est un entier qui va être affecté à l'int "commande".

Quelques dernières corrections à faire

-L'affichage des accents ne se fait pas, à réécrire sans accent
-Le message d'erreur du jeu perdu ne s'affiche pas (celui du jeu gagné s'affiche)
-Lors de la toute première pression d'une flèche, il n'y a pas de réaction (bug mineur)
-Mettre le score en évidence avec un style "inversé" (noir sur blanc)

Pour le dernier point, on utilisera la commande :

```
attron(A_REVERSE);
printw(" SCORE : %d ", plateau[4][0]) ;
attroff(A_REVERSE);
```

Et le message de fin de jeu a été codé par une nouvelle fonction annexe dans le main.cpp. Ainsi, le message reste affiché même lorsque l'on appuie sur une touche jusqu'à ce que l'on restart la partie.