

TP2 - Mini-projet Tetris simplifié

1 - Objectifs

L'objectif de ce TP, ainsi que du prochain, est de réaliser une version simplifiée du jeu Tetris. Si vous ne connaissez pas ce jeu, vous pouvez en apprendre les principes à l'adresse :

<http://fr.wikipedia.org/wiki/Tetris>

Il est important de LIRE ATTENTIVEMENT ET EN ENTIER CE DOCUMENT AVANT DE COMMENCER A CODER!!

L'idée ici est de faire un jeu Tetris console, qui codera toute la logique du jeu, sans les aspects graphiques ni temps réel. Ces deux derniers aspects seront traités dans le TP suivant (**TP3 - Mini-tetris graphique avec GTK**). Ici, l'objectif est de vous faire travailler sur les structures de données, avec d'abord une représentation **tableau à 2 dimensions**, puis ensuite une représentation **liste**, pour stocker la grille du tetris.

Plusieurs contraintes sur la structure de votre programme vous sont imposées.

Pour ce TP, vous devrez remettre au moins deux fichiers : tp2-[votre nom]-tableau.c et tp2-[votre nom]-liste.c. Les points suivants seront évalués :

- La lisibilité de votre programme (choix pertinent pour les noms de variables, indentation, etc.),
- Chaque fichier devra comporter un commentaire au début avec vos nom, prénom, filière, intitulé du cours et numéro de TP,
- Votre code devra compiler sans erreurs ni warnings,
- Toute mémoire allouée devra être libérée.
- Ce TP peut être fait par binôme (mais mettez les deux noms !).
- Groupe 1: Vous m'enverrez une première version de votre TP à la fin de la séance TP du **12 octobre 2016**, puis la version finale avant le dimanche **18 octobre 2016 minuit**, via [TPLab](#). Il faudra une archive nommée TP2-[votre ou vos nom(s)] contenant tous les fichiers sources, entêtes, makefile.
- Groupe 2: Vous m'enverrez une première version de votre TP à la fin de la séance TP du **16 octobre 2016**, puis la version finale avant le jeudi **22 octobre 2016 minuit**, via [TPLab](#). Il faudra une archive nommée TP2-[votre ou vos nom(s)] contenant tous les fichiers sources, entêtes, makefile.
- En plus des fichiers C, vous ajouterez un fichier README textuel dans lequel vous préciserez l'**état d'avancement** de votre TP: quelles questions sont traitées et fonctionnent, quelles questions sont traitées partiellement, quelles questions n'ont pas été abordées.

2 - Une version simplifiée du jeu Tetris

Dans le cadre de ce TP, le jeu sera implémenté uniquement en texte et son déroulement se fera entièrement dans le terminal. Pour cette partie, vous écrirez tout dans un fichier tp2-[votre nom]-tableau.c

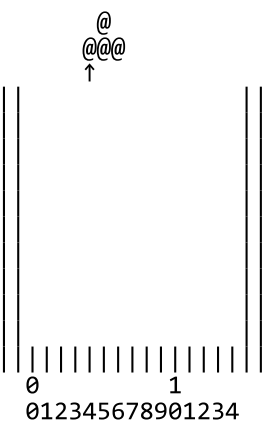
Tout d'abord nous allons simplifier grandement le déroulement du jeu.

1. L'interactivité du jeu sera réduite au minimum :
 - Le joueur ne peut pas tourner les pièces, il ne fait que choisir dans quelle colonne la pièce est déposée.
 - La lecture du clavier étant effectuée à l'aide de la commande "scanf" aucun délai n'est imposé au joueur.

```
* int colonne;  
* scanf( "%d", &colonne );  
*
```

- Lorsqu'un joueur a choisi la colonne dans laquelle faire tomber la pièce, celle-ci est directement ajoutée à la position la plus basse qu'elle peut atteindre.

Voici un exemple :

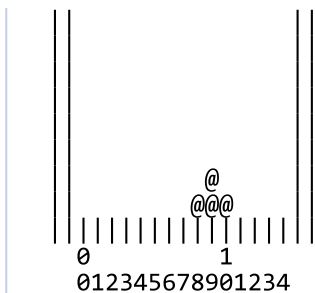


```
Dans quelle colonne placer cette piece?  
>
```

Si le joueur choisit, par exemple, la colonne 8 alors la pièce est insérée de manière à ce que sa colonne la plus à gauche (celle indiquée par une flèche dans l'exemple) soit dans la colonne #8.

```
Dans quelle colonne placer cette piece?  
> 8
```





Dans quelle colonne placer cette piece?
>

Par convention, la grille de jeu est modélisée de bas en haut, c'est-à-dire que la ligne 0 est en bas, la ligne 1 est juste au-dessus, etc. Attention, pour l'affichage sur la console, il faudra donc partir de la dernière ligne et afficher jusqu'à la première ligne.

1. Dans un **premier temps**, les pièces sont dessinées de manière à ce que la ligne la plus basse soit continue et au moins aussi large que toutes la plus large.

	1						
	1		R			LL	Z
@	1	\$\$	R		@@@	L	ZZ
@@@	1	\$\$	RR ...		@	L	Z
pièces valides				pièces non-valides			

Comme nous le verrons plus tard, cette contrainte simplifie grandement le positionnement des pièces.

Remarques :

- Les bords de la grilles de jeu sont représentés par le symbole '|' (pipe)
- Les pièces sont représentés à l'aide de n'importe quels caractères autre qu'un espace (' ') ou un pipe ('|').

3 - Réalisation du programme en C

3.1 - Constantes

Déclarez 4 constantes avec la directive `#define` :

- HAUTEUR : le nombre de lignes de l'espace de jeu. On la fixe à 10.
- LARGEUR : le nombre de colonnes de l'espace de jeu. On la fixe à 15.
- NB_PIECES : le nombre de pièces différentes. À fixer en fonction du nombre de pièces que vous déciderez d'inclure.
- HAUTEUR_MAX_DES_PIECES : le nombre maximum de lignes qui peut occuper une pièce. On fixe cette valeur à 4.

3.2 - Nouveaux types

- **Piece** : une struct contenant les champs suivants
 - **hauteur** : le nombre de lignes qu'occupe la pièce.
 - **largeur** : le nombre de colonnes qu'occupe la pièce.
 - **forme** : un tableau de chaînes de caractères, permettant de dessiner la pièce. Ce sera un tableau de `HAUTEUR_MAX_DES_PIECES` cases, chaque case étant de type `char*`, qui pointe donc vers une chaîne de caractères.
- **Grille** : Un tableau bidimensionnel de caractères qui représente l'espace de jeu.

3.3 - Initialisation et affichage de la grille

- Écrivez une procédure `initialiseGrille` qui remplit une grille de jeu passé en paramètre avec le caractère ' ' (espace).
- Écrivez une procédure `lireCase` qui, pour une grille, un numéro de ligne et un numéro de colonne passés en paramètre, retourne le contenu de la case correspondante de cette grille. Vous devez toujours utiliser cette procédure lorsque vous voulez lire le contenu d'une case la grille de jeu. Ajoutez dans cette procédure un test vérifiant que la ligne et la colonne sont valides. Si tel n'est pas le cas, une erreur doit être affichée.
 - Écrivez une procédure `afficheGrille` afin d'afficher le contenu d'une grille donnée en entrée en l'encadrant à gauche, à droite et en dessous avec le caractère '|' (pipe). Ajoutez également des nombres qui indiquent le numéro de chacune des colonnes (voir l'exemple plus haut).
- Écrivez une procédure `main` qui vous permet de tester le fonctionnement des deux procédures précédentes.

TESTEZ VOTRE PROGRAMME À CHAQUE ÉTAPE DE SON DÉVELOPPEMENT!!

3.4 - Génération et affichage des pièces

- Écrivez une procédure `générerPièces` qui initialise un tableau de pièce avec chacune des pièces qui apparaîtront dans le jeu. Les pièces sont écrites une par une "à la main" dans cette fonction. Voici un exemple d'initialisation d'une pièce :

```
tabPiece[i].hauteur = 2;
tabPiece[i].largeur = 3;
tabPiece[i].forme[1] = "@";
tabPiece[i].forme[0] = "@@@";
```

Il est important que chacune des chaînes de caractères du tableau forme aient exactement la même taille, cette taille étant la valeur donnée au champs largeur (3 dans cet exemple). Votre jeu doit fournir au moins trois pièces différentes, à vous de décider lesquelles. Notez que comme le joueur ne peut pas tourner les pièces, les quatre pièces suivantes sont considérées comme étant différentes :

```

#           #
#           #           #           #
##          ##          ###          ###

```

- Écrivez une procédure `affichePiece` qui affiche une pièce passée en paramètre et ajoute une flèche (ou tout autre symbole) sous la colonne la plus à gauche de manière à indiquer au joueur où sera inséré la pièce.



3.5 - Écriture dans la grille

- Écrivez une procédure `ecrireCase` qui, pour une grille, un numéro de ligne et un numéro de colonne passés en paramètre, inscrit dans la case correspondante de la grille de jeu un caractère également spécifié en paramètre. Vous devez toujours utiliser cette procédure pour écrire dans la grille de jeu. Il est là aussi fortement conseillé de tester la validité des numéros de ligne et colonne et d'afficher une erreur dans le cas contraire.

Lorsqu'un joueur décide de placer une pièce dans une colonne donnée, il faut être en mesure de déterminer à quelle hauteur la pièce sera déposée. On a imposé une forme particulière aux pièces de manière à simplifier cette étape. Comme la ligne la plus basse d'une pièce est forcément la plus large, il suffit de déterminer, pour chacune des colonnes que va occuper cette pièce, quelle est hauteur de la plus haute case occupée.

- Écrivez une procédure `hauteurMax` qui, étant donné une grille et un intervalle de colonnes, retourne la hauteur maximal où se trouve une case occupée et retourne -1 si aucune case n'est occupée. Cette procédure sera mise à jour dans un **deuxième temps** pour prendre en compte des pièces plus complexes (voir plus loin).
- Écrivez une procédure `ecrirePiece` qui reçoit en paramètre une grille, une pièce, un numéro de colonne ainsi qu'une hauteur et ajoute cette pièce à la grille de manière à ce que la colonne la plus à gauche de la pièce corresponde au numéro de colonne spécifiée. Faites bien attention à ne **rien** écrire là ou la pièce est vide (ie. un espace). Ce sera important pour la section **3.9 - (OPTIONNEL) Pièces générales**.

3.6 - Lecture des entrées

- Écrivez une procédure `pieceAleatoire` qui choisit une pièce au hasard parmi celles que vous avez définies. Pour choisir un nombre aléatoirement dans l'ensemble {0,1,2,...,n-1} on peut utiliser la commande suivante :

```
#include <stdlib.h>
...
int alea = (int)((((double)random())/((double)RAND_MAX)) * (NB_PIECES));
```

- Modifiez la méthode `main` en y ajoutant une boucle principale de manière à :
 - Afficher une pièce choisie aléatoirement,
 - Affiche la grille de jeu,
 - Demande à l'utilisateur d'entrer le numéro de la colonne où il veut mettre la pièce,
 - Ajoute la pièce à la grille de jeu.
 - Répéter les étapes précédentes jusqu'à ce que l'utilisateur entre la valeur -1 indiquant qu'il souhaite quitter le programme.

3.7 - Détection de la fin de la partie

- Modifiez votre code de manière à ce qu'avant de faire une appel à `ecrirePiece` votre programme détecte si la pièce va dépasser de la grille de jeu. Si tel est le cas, on affiche un message informant le joueur qu'il a perdu la partie ainsi que le nombre de pièces qu'il a réussi à placer. La grille est alors réinitialisée et une nouvelle partie démarre.

3.8 - Effacement de lignes

- Écrivez une procédure `supprimerLigne` efface le contenu d'une ligne dont le numéro est passé en paramètre et fait descendre toutes celles au-dessus. Plus précisément, lorsqu'une ligne est supprimée, le contenu de chacune des lignes au dessus de celle-ci doit être recopié dans la ligne d'en dessous et la ligne la plus haute de la grille de jeu est remplacée par une ligne vide.
- Écrivez une procédure `nettoyer` qui supprime toutes les lignes ne contenant aucune case vide.
- Modifiez votre code de manière à ce qu'à chaque fois que le joueur place une pièce, un appel à `nettoyer` soit effectué.

Vous devriez avoir maintenant une version jouable. Malheureusement, les pièces étant toutes plus larges à la base, vous allez voir qu'il est difficile de gagner à ce Tetris.

3.9 - (OPTIONNEL) Pièces générales

On autorise maintenant le type de pièces ci-dessous:



Il n'est plus suffisant d'utiliser `hauteurMax` pour détecter la position où va tomber la pièce. On peut juste dire que `hauteurMax` donne la "pire" hauteur possible, mais potentiellement la pièce peut aller plus bas. Ecrivez donc la fonction

```
int hauteurExacte( Grille g, int col_gauche, Piece* piece );
```

qui vous retourne cette hauteur exacte. Substituez ensuite `hauteurExacte` à `hauteurMax` dans le code. Si la procédure `ecrirePiece` est correctement écrite, alors cela devrait fonctionner du premier coup.

4 - Implémentation à l'aide d'une liste chaînée

Cependant, on aimerait ne pas avoir à recopier la grille case par case lorsqu'une ligne est supprimée. Pour cela, nous allons remplacer le tableau servant à représenter la grille de jeu par une liste doublement chaînée.

- Commencez par effectuer une copie de sauvegarde de votre code. Appelez ce fichier

```
tp2-[votre nom]-tableau.c
```

Avant de modifier votre code, récupérez le code suivant, compilez-le et exécutez-le.

Fichier Liste.h

```

ifndef _LISTE_H
#define _LISTE_H

typedef double Elem; /* Vous changerez après ce type lorsque vous l'utiliserez pour le tetris */
struct SCellule {
    Elem val;
    struct SCellule* pred;
    struct SCellule* succ;
};
typedef struct SCellule Cellule;
typedef Cellule* Adr;
typedef Cellule Liste;
/* Ici, la liste vide est une liste avec un élément (non utilisé). */

/* alloue dynamiquement une liste et retourne son adresse */
extern Liste* Liste_creer();
/* initialise correctement la liste donnée en paramètre, comme si elle était vide. */
extern void Liste_init( Liste* L );
/* détruit tous les éléments stockés dans la liste L. La liste est vide après.*/
extern void Liste_termine( Liste* L );
/* détruit tous les éléments stockés dans la liste L, et libère l'espace mémoire de la liste. */
extern void Liste_detruire( Liste* L );
/* retourne l'adresse du premier élément. */
extern Adr Liste_debut( Liste* L );
/* retourne l'adresse après le dernier élément. */
extern Adr Liste_fin( Liste* L );
/* passe à l'élément suivant. */
extern Adr Liste_suivant( Liste* L, Adr A );
/* passe à l'élément précédent. */
extern Adr Liste_precedent( Liste* L, Adr A );
/* insère devant l'élément A un nouvel élément de valeur v dans L. */
extern Adr Liste_insere( Liste* L, Adr A, Elem v );
/* supprime l'élément A dans L. */
extern void Liste_supprime( Liste* L, Adr A );
/* retourne la valeur stockée dans l'élément A de la liste L. */
extern Elem Liste_valeur( Liste* L, Adr A );
/* modifie la valeur stockée dans l'élément A de la liste L, en lui assignant la valeur v. */
extern void Liste_modifie( Liste* L, Adr A, Elem v );

#endif
```

Fichier Liste.c

```

#include <stdlib.h>
#include "Liste.h"

Liste* Liste_creer()
{
    Liste* L = (Liste*) malloc( sizeof( Liste ) );
    Liste_init( L );
    return L;
}

void Liste_init( Liste* L )
{
    L->succ = L;
    L->pred = L;
}

void Liste_termine( Liste* L )
{
    while ( Liste_debut( L ) != Liste_fin( L ) )
        Liste_supprime( L, Liste_debut( L ) );
}

void Liste_detruire( Liste* L )
{
    Liste_termine( L );
}

Adr Liste_debut( Liste* L )
{
    return L->succ;
}

Adr Liste_fin( Liste* L )
{
    return L;
}

Adr Liste_suivant( Liste* L, Adr A )
{
    return A->succ;
}

Adr Liste_precedent( Liste* L, Adr A )
{
    return A->pred;
}

Adr Liste_insere( Liste* L, Adr A, Elem v )
{
    Adr ncell = (Adr) malloc( sizeof( Cellule ) );
    ncell->val = v;
    ncell->succ = A;
    return ncell;
}

void Liste_supprime( Liste* L, Adr A )
```

```
{
    A->pred->succ = A->succ;
    A->succ->pred = A->pred;
}

Elem Liste_valeur( Liste* L, Adr A )
{
    return A->val;
}

void Liste_modifie( Liste* L, Adr A, Elem v )
{
    A->val = v;
}
```

Fichier : test-Liste.c

```
#include <stdio.h>
#include "Liste.h"

void affiche( Liste* L )
{
    Adr A;
    for ( A = Liste_debut( L ); A != Liste_fin( L );
          A = Liste_suivant( L, A ) )
        printf( "%f ", Liste_valeur( L, A ) );
    printf( "\n" );
}

int main( void )
{
    Liste* L = Liste_creer( );
    Adr A = Liste_debut( L );
    double x = 1.0;
    while ( x < 100000.0 )
    {
        A = Liste_insere( L, A, x );
        A = Liste_suivant( L, A );
        x = 1.5*x;
    }
    affiche( L );
    Liste_detruire( L );
    return 0;
}
```

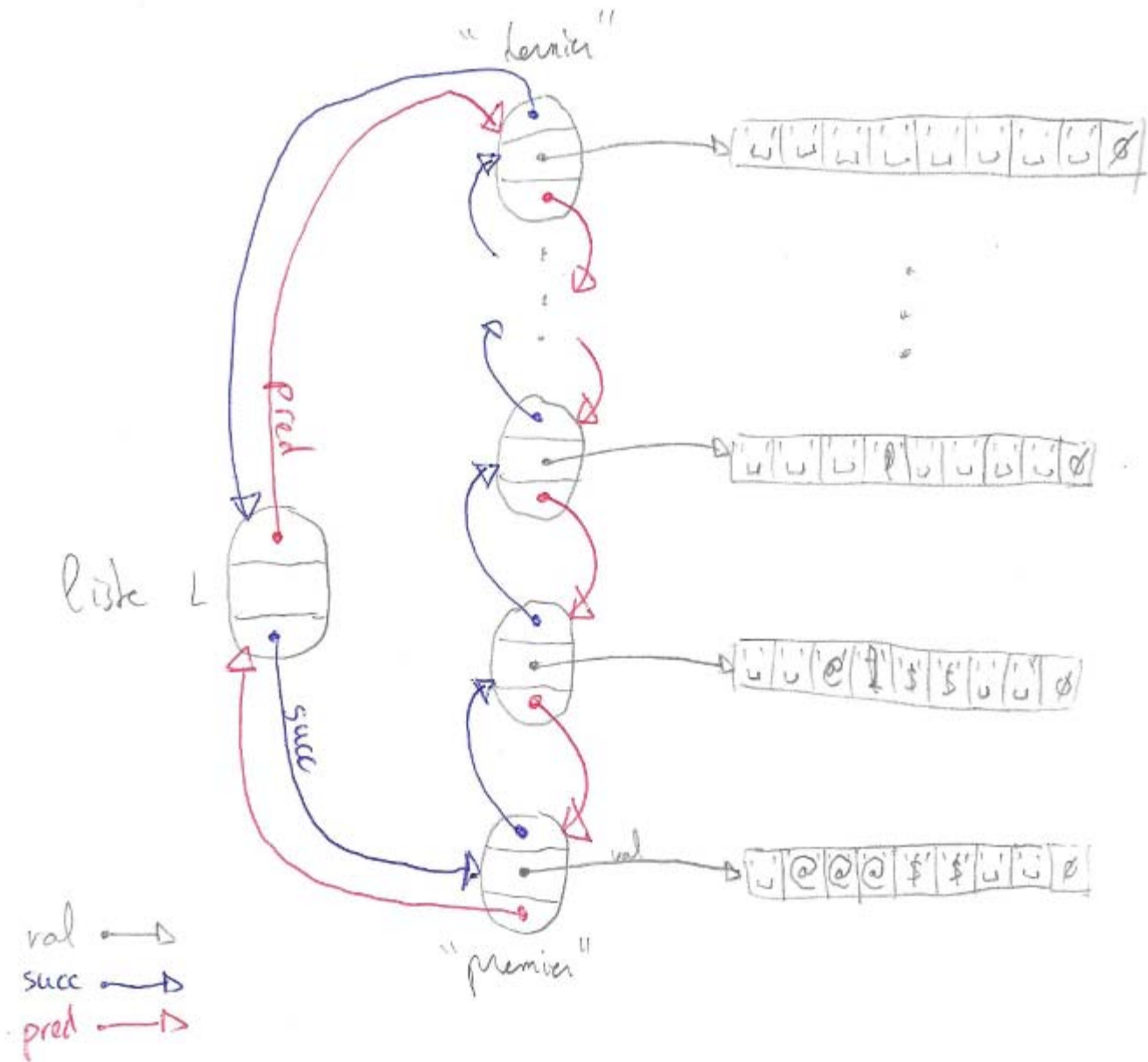
On constate que, contrairement à ce que l'on espérerait, la commande `affiche(L)` n'affiche rien du tout.

- Utilisez un débogueur (par exemple : ddd), afin de corriger cette implémentation de liste chaînées.
- Assurez-vous que ces listes libèrent toute la mémoire allouée en utilisant l'outil `valgrind` de la manière suivante :

```
valgrind --leak-check=full ./test-Liste
```

Lorsque vos listes fonctionnent correctement, vous pouvez les adapter afin de les utiliser pour représenter efficacement la grille de jeu de votre Tetris. Chaque noeud de votre liste représentera une ligne du jeu tetris, la première ligne (celle du bas) étant la première cellule utile de votre liste chaînée. Supprimer une ligne se fera donc en supprimant la cellule correspondante et vous devrez rajouter une ligne composée d'espaces en fin de liste. Voilà à quoi ressemble votre liste pour représenter le tetris ci-contre.

```
||      ||
...
||  1  ||
|| @1$$ ||
|| @@@$$ ||
```

- Définissez un type pour représenter chacune des lignes de la grille de jeu.
- Redéfinissez le type Grille de manière à remplacer le tableau par une liste chaînée dont chaque cellule représente une ligne de la grille.

À ce stade, la compilation de votre programme échouera. C'est normal. Il faut maintenant modifier les procédures qui manipulent la grille de jeu.

- Afin de pouvoir compiler votre code, commentez le corps des procédures `ecrireCase` et `lireCase`. Il peut s'avérer pratique d'ajouter un `return` bidon (ex : `return ' ';`) à la procédure `lireCase`.

Votre programme devrait maintenant compiler à nouveau tout en restant inutilisable.

- Écrivez une procédure `construireGrille` qui construit la grille de jeu.
- Écrivez une procédure `détruireGrille` qui libère la mémoire occupée par la grille.
- Modifier le corps des procédures `lireCase` et `ecrireCase` afin de les adapter à la nouvelle structure de données employée pour la grille de jeu.

Votre programme devrait maintenant être redevenu fonctionnel.

- Modifier le corps de la procédure `supprimerLigne` de manière à ne plus recopier la grille case par case mais plutôt en supprimant l'élément approprié de la liste chaînée et en ajoutant une nouvelle ligne vide en bout de liste.
- Renommez votre fichier source

```
tp2-[votre nom]-liste.c
```

5 - Améliorations optionnelles

- Si vous avez terminé les étapes précédentes et que tout fonctionne bien, vous pouvez améliorer votre code en modifiant les procédures `afficherGrille`, `initialiser` et `hauteurMax` afin de remplacer les appels à `ecrireCase`, `lireCase` par une utilisation séquentielle de la liste chaînée.
- Vous pouvez rajouter des couleurs sur votre console, voir par exemple <http://fr.openclassrooms.com/informatique/cours/des-couleurs-dans-la-console-linux>
- Vous pouvez apporter tout autre modification qui rendra votre programme plus semblable au vrai Tetris ;-): toutes les pièces, score (en fonction du nombre de lignes supprimé en même temps). L'aspect temps réel sera abordé dans le TP suivant.

6 - Notes

- Pour tester plus facilement votre tetris, mettez la pièce verticale dans la liste des pièces possibles.
- Dans la version liste du Tetris, une erreur commune est d'initialiser chaque ligne par une affectation à la chaîne " _____ " (où les ' ' sont des espaces). Le problème est que c'est une chaîne constante, donc non modifiable. Il faut donc bien allouer **dynamiquement** une chaîne de la bonne longueur et la remplir d'espaces.