LC Développe ent en C

Projet De ineur

⋆ ⋆ ⋆

Le ut de ce projet est de progra er un de ineur en ode te te. Pour cela on utilisera les structures suivantes ·

```
#define DEJAVU 1
#define PASVU  0

struct parcelle{
    int mine;    /* 1 s'il y a une mine, 0 sinon */
    int status;  /* DEJAVU si deja vu, PASVU sinon */
};
typedef struct parcelle parcelle;

typedef struct str_champs{
    int nb_erreurs_restant;
    int nb_lignes;        /* nombre de lignes   */
    int nb_colonnes;      /* nombre de colonnes */
    parcelle *tab;        /* pointeur vers un tableau de     */
                          /* nb_lignes*nb_colonnes parcelles */

} str_champs;
typedef str_champs *champs;
```

On donnera au joueur le droit à un no re li ité d'erreurs. Si le cha ps `nb_erreurs_restant` est 0 et que le joueur passe sur une parcelle ine le jeu se ter ine i édiate ent avec un essage approprié. Par contre si `nb_erreurs_restant >0` le passage sur une parcelle ine di inuera ce co pteur.

ous écrirez les fonctions suivantes ·

( ) La fonction

```
p_champs init(int nb_lignes, int nb_colonnes, int nb_mines)
```

qui construit et retourne un cha ps de taille `nb_lignes` sur `nb_colonnes` contenant `nb_mines` ines. Les ines devront être placées aléatoire ent.

pouf srand(fsfrand(/ pouf id(/

( ) la fonction

```
int nb_mines_voisins(p_champs C, int i, int j)
```

qui retourne le nombre de mines contenues dans les parcelles voisines de la parcelle `(i,j)`. (Chaque parcelle a au plus 8 voisins : nord, nord-est, est, sud-est, etc.).

( ) la fonction

```
void affichage(p_champs C)
```

qui affiche l'état du champs `C`.

Les parcelles non visitées sont marquées par le symbole `x`.

Les parcelles visitées non minées sont marquées par le nombre de voisins qui contiennent une mine.

Les parcelles minées visitées sont marquées par la lettre `M`.

Par exemple pour un champs de 13×18 après trois coups joués et lorsque le joueur n'a le droit qu'à une erreur en début de partie, l'affichage prendra la forme suivante :

```
      0                   1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
     -----------------------------------
0 0 | x x x x x x x x x x x x x x x x x x
  1 | x x x x x x x x x x x x x x x x x x
  2 | x x x x x x x x x x x x x x x x x x
  3 | x x x x x x x x x 2 x x x x x x x x
  4 | x x x x x x x x x x x x x x x x x x
  5 | x x x x x x x x x x x x x x x x x x
  6 | x x x x x x x x x x x x x x x M x x
  7 | x x x x x x x x x x x x x x x x x x
  8 | x x x x x x x x x x x x x x x x x x
  9 | x x x x x x x x x x x x x x x x x x
1 0 | x x x x x 3 x x x x x x x x x x x x
  1 | x x x x x x x x x x x x x x x x x x
  2 | x x x x x x x x x x x x x x x x x x

  Il reste 154 parcelle(s) non minee(s) a trouver.
  Tu as encore droit a 0 erreur(s).
```

( ) Le pointeur `tab` dans la structure `str_champs` est un pointeur vers un tableau à une dimension. Il faut donc prévoir les fonctions de conversion suivantes :

```
int indice(champs C, int i, int j)
```

qui retourne l'indice dans ce tableau pour une parcelle dont les coordonnées sont `(i,j)` et

```
int getX(champs C, int k)
int getY(champs C, int k)
```

qui retournent respectivement les numéros de ligne et de colonne correspondant à l'élément de l'indice `k` dans le tableau.

( ) La fonction

```
int sauvegarder(char *fichier, p_champs C)
```

qui permet de sauvegarder l'état du jeu dans un fichier. Cette fonction retourne 1 en cas de succès et 0 en cas d'échec.

(6) La fonction

```
p_champs lire(char *fichier)
```

qui lit l'état du démineur à partir d'un fichier. Cette fonction retourne `NULL` en cas de problème (il possible de lire le fichier le contenu du fichier est incorrect ou le format est incompatible avec le format utilisé par la fonction `sauvegarder`).

( ) La fonction

```
int joueur(champs C, int i, int j)
```

qui permet au joueur de jouer un coup sur la parcelle de position (i, j). Cette fonction retourne :

- 0 si le coup est réussi (si la parcelle n'a jamais été visitée et ne contient pas de mine)
- 1 si la parcelle (i, j) n'a jamais été visitée mais contient une mine
- 2 si la parcelle a déjà été visitée auparavant
- -1 si la parcelle (i, j) n'existe pas sur le champs

Enfin vous écrirez la fonction `main` de la façon suivante :
Tant que la partie n'est pas terminée on affiche à chaque coup l'état du jeu et le menu suivant :

```
Q ou q pour quitter
S ou s pour sauvegarder
Coordonnees du point:
```

Si le joueur choisi la commande `S` alors on demandera le nom du fichier où sauvegarder la partie. Si ce fichier existe on demandera une confirmation avant de sauvegarder. Si la sauvegarde échoue on en informera le joueur puis on lui proposera soit de corriger le nom de fichier soit de revenir vers le menu.

Vous découperez les sources en plusieurs fichiers de la façon la plus appropriée. Il faut que votre programme soit accompagné d'un Makefile permettant une compilation par un simple `make`. La commande `make clean` effacera les fichiers de sauvegarde d'essais `make cleanall` effacera en plus le fichier exécutable et les fichiers `*.o`.

Le jeu du démineur pourra commencer de plusieurs manières :
- `demineur -f fichier` qui initialise le jeu à partir d'un fichier de sauvegarde
- `demineur l c m` qui initialise le jeu avec `l` lignes `c` colonnes `m` mines et 0 erreurs possibles
- `demineur l c m e` qui initialise le jeu avec `l` lignes `c` colonnes `m` mines et `e` erreurs possibles.

Enfin vous prévoierez dans votre programme une entrée secrète : après chaque tour la commande `A` *non répertoriée dans le menu* affichera toutes les mines cachées dans le champs.

Vous pourrez ajouter d'autres fonctions de votre choix. On préfèrera des petites fonctions bien ciblées à des grandes fonctions difficiles à comprendre et à maintenir. Les fichiers sources devront être commentés convenablement. Il sera demandé pour chaque fonction une description claire de ce qu'elle fait ainsi que des commentaires qui faciliteront la lecture et la compréhension mais vous éviterez les commentaires trop détaillés.

Chaque fichier source commencera par un commentaire donnant les noms et prénoms des auteurs du projet.

En ce qui concerne la notation la clarté et la qualité de votre code compteront le plus dans la note du projet. Quelques conseils un programme bien structuré et bien indenté est plus compréhensible de plus une variable bien nommée est souvent plus pertinente qu'un long commentaire.
Le projet (fichiers sources Makefile) devra être envoyé à votre chargé de TD/TP (voir la page du cours pour les adresses email) sous forme d'un archive `*.tar.gz`

Pour un binôme composé de XXX et YYY l'archive doit porter le nom `XXX_YYY.tar.gz`.
Pour archiver vous pourrez utiliser la commande :

On en déduit que le projet sera réalisé en binômes.

`tar czvf XXX_YYY.tar.gz` dans le répertoire parent de votre projet.
La décompression de votre archive à l'aide la commande `tar xzvf XXX_YYY.tar.gz` devra produire le répertoire `XXX_YYY` contenant les sources (fichier `*.c *.h`) et votre Makefile.
Et finalement la date limite de la remise du projet **le 11 avril 2008 à 23h**.