1 Le démineur

Le démineur est un jeu de réflexion sur un plateau de jeu à deux dimensions de forme rectangulaire, qui se joue à un seul joueur. Des mines sont cachées sous certaines cases du plateau. Un tour de jeu consiste à révéler une case du plateau. Si le joueur révèle une mine, il a perdu la partie. Si toutes les cases qui ne sont pas des mines ont été révélées, le joueur a gagné.

Lorsqu'une case est révélée, si elle n'est pas une mine, le nombre de mines adjacentes à cette case est donné. De plus, s'il n'y a aucune mine autour de cette case, toutes les cases voisines sont également révélées. Cette opération est récursive : l'action de révéler une case peut ainsi en révéler beaucoup d'autres.

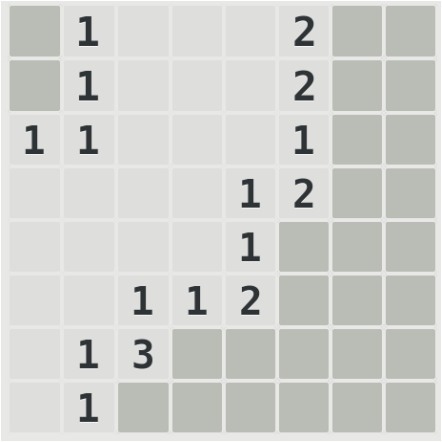

FIGURE 1 – Capture d'écran du programme gnome-mines, une implémentation libre du jeu. Les cases fermées sont en gris foncé, les cases ouvertes sont en gris clair et comportent le nombre de mines dans le voisinage de la case. Pour ne pas surcharger l'affichage, les 0 ne sont pas affichés.

2 Modélisation

On peut modéliser le plateau de jeu par trois tableaux à deux dimensions ($largeur \times hauteur$). Le premier tableau **mines** contient des booléens vrais si et seulement si les cases correspondantes sont des mines. Le second tableau **neigh** contient des entiers, qui correspondent au nombre de mines adjacentes à chaque case. Enfin, un dernier tableau **open** stocke dans des booléens si les cases ont été ouvertes (révélées) ou non.

On va utiliser la première dimension du tableau pour la hauteur et la seconde pour la largeur, afin que les cases soient contigues lorqu'elles sont parcourues dans l'ordre d'affichage. Ainsi, le code suivant accède à la case de coordonnées (x, y) du tableau mines, avec $x \in \mathbb{N}, 0 \leq x \leq largeur$ et $y \in \mathbb{N}, 0 \leq y \leq hauteur$.

```
mines[y][x];
```

3 Un premier programme

Écrivez une base de code, qui inclut les fichiers de la bibliothèque standard dont vous avez besoin (laissez un peu de place pour pouvoir éventuellement y rajouter des fichiers plus tard) et définit une fonction **main**. Votre fonction **main** doit demander la saisie des dimensions du plateau (la largeur **w** doit être dans $[1, 50]$ et la hauteur **h** dans $[1, 20]$), ainsi qu'un nombre de mines (dans $[1, w \times h]$), puis définit les trois tableaux **mines**, **neigh** et **open**.

4 Initialisation simples

```
void initialise_mines(int width, int height, bool ** mines,
                      int nb_mines);
void initialise_open(int width, int height, bool ** open);
```

Implémentez les fonctions **initialise_mines** et **initialise_open**, qui initialisent respectivement les tableaux **mines** et **open**.

La première fonction doit placer **nb_mines** à des coordonnées aléatoires valides du plateau. On suppose que **nb_mines** est dans $[1, w \times h]$. Attention, il faut qu'il y ait exactement **nb_mines** dans le plateau après l'appel de cette fonction.

La seconde fonction doit fermer toutes les cases du plateau.

5 Nombre de mines et voisinage

```
bool is_coord_ok(int width, int height, int x, int y);
int nb_adjacent_mines(int width, int height, const bool ** mines,
                      int x, int y);
void compute_neigh(int width, int height, const bool ** mines,
                   int ** neigh);
```

Implémentez ces trois fonctions.

La fonction **is_coord_ok** doit renvoyer vrai si et seulement si les coordonnées (x, y) sont des coordonnées valides du plateau de largeur *width* et de hauteur *height*.

La fonction **nb_adjacent_mines** doit calculer et renvoyer le nombre de mines dans le voisinage de la case de coordonnées (x, y). On suppose que les coordonnées (x, y) sont valides. Le voisinage d'une case est composé de 8 autres cases : celles autour de la case (en comprenant les diagonales).

Enfin, la fonction **compute_neigh** doit initialiser le tableau **neigh**, de telle sorte que chaque case de coordonnées (x, y) du tableau **neigh** contienne le nombre de mines adjacentes à cette case.

6 Affichage du plateau

```
void display_board(int width, int height, const bool ** mines,
                   const bool ** open, const int ** neigh)
```

Implémentez la fonction **display_board**, chargée d'afficher un plateau de jeu. Vous devez pour cela afficher chaque case non ouverte par un ■. Les cases ouvertes qui sont des mines doivent être affichées par des ⋆. Les cases ouvertes qui n'ont aucune mine adjacente par des ∘, et enfin, les cases ouvertes ayant un nombre de mines adjacentes non nul doivent afficher ce nombre (un chiffre entre 1 et 8).

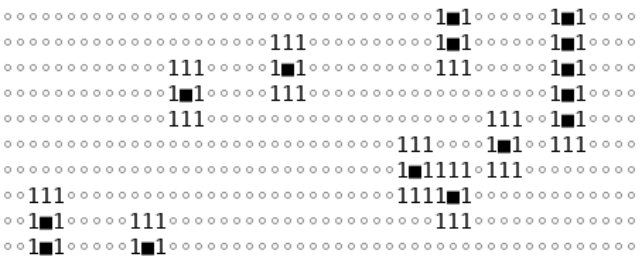

FIGURE 2 – Une capture d'écran du type d'affichage demandé

7 Demandons un tour de jeu

```
bool is_move_ok(int width, int height, const bool ** open,
                int x, int y);
void ask_move(int width, int height, const bool ** open,
              int * x, int * y);
```

Implémentez ces deux fonctions.

La fonction **is_move_ok** doit renvoyer si jouer aux coordonnées (x, y) du plateau est un coup valide ou non. Pour que le coup soit valide, il faut que les coordonnées (x, y) soient valides et que la case de coordonnées (x, y) ne soit pas ouverte.

La fonction **ask_move** demande au joueur où il souhaite jouer tant que le joueur n'a pas saisi de coup valide. Les paramètres x et y de cette fonction sont des paramètres de sortie, ils permettent de renvoyer où le joueur souhaite jouer.

8 Inondation du plateau

```
void propagate_open_cells(int width, int height, bool ** open,
                          const int ** neigh)
```

Lorsqu'un joueur joue un coup, il ouvre une case. Si cette case n'est pas une mine et n'a pas de mines dans son voisinage, toutes les cases adjacentes sont également ouvertes. Il en va ensuite de même pour les cases nouvellement ouvertes... Ainsi, de plus en plus de cases peuvent être ouvertes jusqu'à saturation du plateau.

Implémentez la fonction **propagate_open_cells**, qui permet d'ouvrir toutes les cases qui doivent l'être dans le plateau grâce à l'algorithme suivant :

```
Faire
  Pour chaque case (x,y) du plateau
    Si la case (x,y) est 1. ouverte, et
                         2. n'est pas une mine, et
                         3. n'a pas de mine adjacente, alors
      Ouvrir les cases voisines non ouvertes de la case (x,y)
Tant qu'il y a eu des cases ouvertes dans le dernier parcours
```

9 Fin de la partie

```
int is_finished(int width, int height, const bool ** mines,
                const bool ** open, int nb_mines);
```

Implémentez la fonction **is_finished** qui doit renvoyer 1 si la partie est gagnée, 2 si la partie est perdue ou 0 si la partie n'est pas terminée. Une partie est perdue si une mine a été ouverte. Une partie est gagnée si toutes les cases n'étant pas des mines ont été ouvertes. Dans les autres cas, la partie n'est pas terminée.

10 Jouons au jeu

```
void play_turn(int width, int height, const bool ** mines,
               bool ** open, const int ** neigh);
void play_the_game(int width, int height, const bool ** mines,
                   bool ** open, const int ** neigh, int nb_mines);
```

Implémentez la fonction **play_turn**, qui joue un tour de jeu :

1. Affichage du plateau de jeu courant,
2. Demande d'un coup au joueur,
3. Découverte de la case du joueur,
4. Découverte éventuellement d'autres cases (via inondation).

Enfin, implémentez la fonction **play_the_game**, qui joue des tours tant que le jeu n'est pas terminé, puis affiche si le joueur a gagné ou perdu.