IUT de Paris - Rives de Seine
Université de Paris

Rapport SDA: Jeu du Démineur
IUT de Paris - Département Informatique

Yanis Seghdau, Elias Ait-Khelifa - Groupe 110

IUT de Paris - Rives de Seine
Université de Paris

Tables des matières

Yanis Segdauh, Elias Ait-Khelifa

Introduction

La création d'un programme est un processus compliqué qui nécessite souvent une réflexion au préalable sur toute sa structure. Grâce au TD/TP et au cours magistraux, nous avons pu réaliser ce programme.
Le langage que nous avons utilisé pour créer ce programme, est le C++.

Le Jeu

Le but du Jeu

Le jeu du démineur est un jeu de hasard et d'analyse logique qui consiste à localiser toutes les mines présentes sur la grille de jeu sans en trouver une. Ce jeu à été créer en 1989, ce jeu vient avec le système d'exploitation Windows.

Comment fonctionne t-il ?

Le jeu est constitué d'une grille où sont cachées les mines. Au fur et à mesure que le jeu avance, certaines cases de la grille sont explorées et des informations apparaissent. Si le joueur découvre une case contenant une mine, le jeu se termine. Si le joueur explore une case proche d'une une mine, on voit apparaître un chiffre indiquant le nombre de mines adjacentes à la case. De plus, pour aider le joueur à mémoriser les endroits qu'il croit minés, il peut les identifier comme des cases minées.
Pour gagner la partie, le joueur doit identifier toutes les cases qui ne sont pas minées

Problématique

Le projet consiste en la création d'un Interpréteur de commandes permettant de produire un problème à partir d'un nombre "x" de ligne, de colonne et de mine ; de produire une grille à partir d'un historique de coups, de déterminer si une partie est gagnée ou pas à partir d'un problème et d'un historique de coup et enfin de produire un nouveau coup à partir d'une grille. Chacune de ces fonctions auront un numéro attribué de 1 à 5 et selon le numéro du début, ces fonctions seront appelées.

Développement

Conception

Dans le cadre de ce projet, nous devions mettre en exécution nos connaissances acquises au cours des TD/TP et CM.
Pour ce faire nous avons utilisé quelques bibliothèques, par exemple : "stdlib.h", assert.h, "random" etc...

Programmation

Pour ce projet, l'utilisation du langage C++ a été choisie : ainsi nous devions nous familiariser avec ce dernier. Nous devions ainsi comprendre son fonctionnement général :

- Les boucles
- L'utilisation de bibliothèques et l'externalisation
- La structuration du programme
- Les fonctions
- etc…

Ce que le programme propose :

Ce programme permet de jouer au célèbre jeu du démineur ! Grâce à une entrée clavier, ce programme permettra de produire un problème avec un nombre de mines, de colonnes et de ligne.
Exemple d'entré clavier : 1657 : 1 est le code d'opération, le 6 est le nombre de lignes, le 5 de colonnes et enfin le 7 le nombre de mines. Le programme propose plusieurs choses en fonction du code d'opéartion allan de 1 à 5.
La 1 cité juste au dessus permettra de créer un problème en prenant en compte le nombre de mines, de colonnes et de lignes.
Le code d'opération 2, vous permettra de produire une grille grace à un historique de coup et d'un problème.
Le code numéro 3 et 4 vous permettra de déterminer si une partie est gagné ou perdue.
Et enfin, le dernier qui est le code d'opération numéro 5 qui malheuresement n'est pas proposé dans notre programme qui permet de produire un nouveau coup.

Bilan du projet

Difficultés rencontrées

Nous avons rencontré plusieurs difficultés dans la réalisation de ce projet. Parmi toutes ces complications, il y a notamment le développement à terme de celui-ci tout en ayant des projets en parallèle (Saé, Semaine de DST…).
Les contraintes liées au projet lui-même sont multiples, nous avons rencontré un très grand problème dès le début. Nous devions réfléchir à la façon d'utiliser le code d'opération jusqu'a ce que l'on trouve une manière toute simple.
La plus grande gêne que nous avons rencontrée est de produire un nouveau coup, nous avons passé quelque semaine sur cela.

Apport du projet

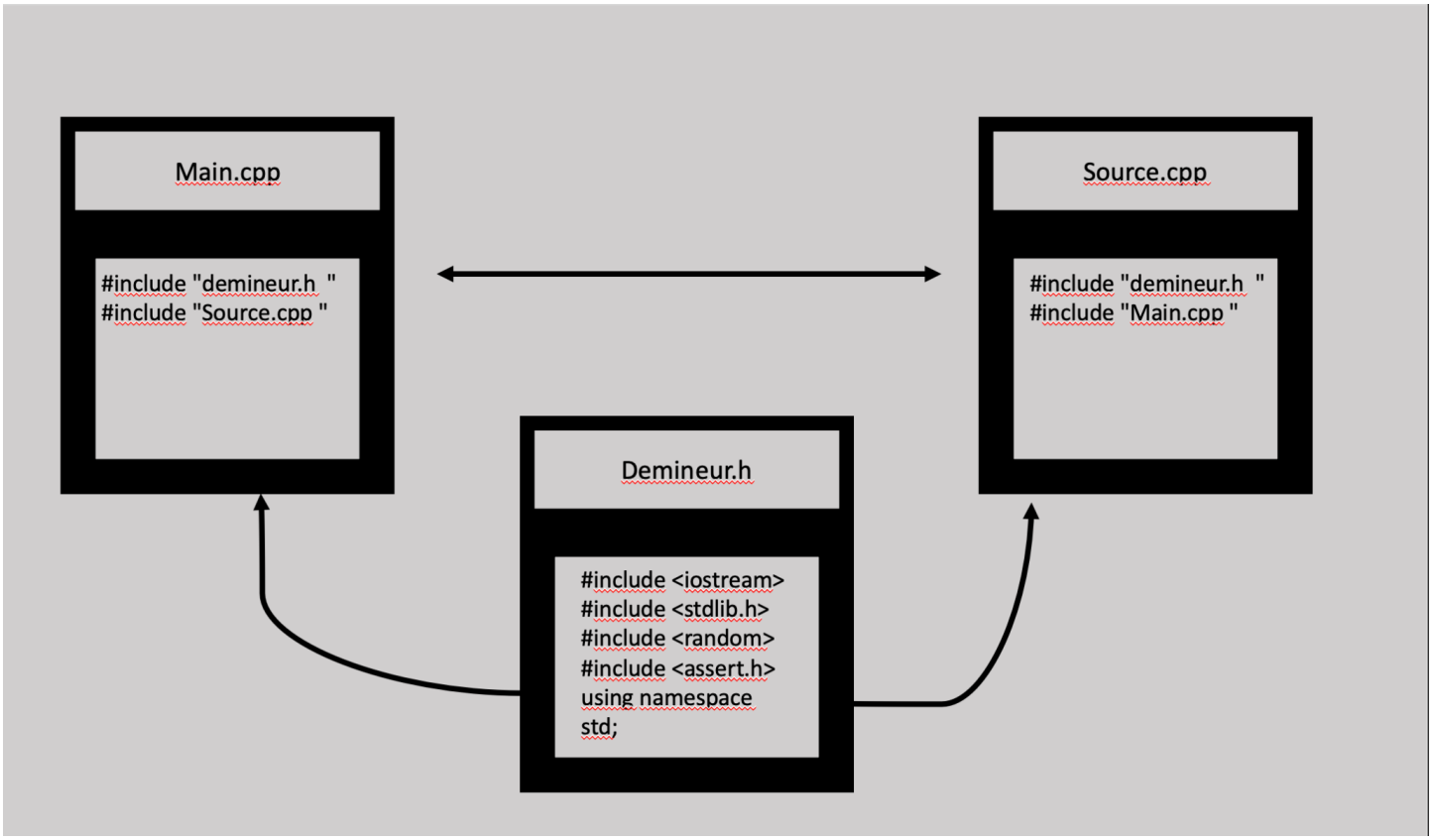
Tout au long de ce projet, nous avons réalisé l'importance de l'organisation dans le travail de groupe.
Nous avons par exemple utilisé Discord pour planifier des réunions/sessions de travail (à l'IUT ou chez nous).
Les appels à distance se faisaient aussi par Discord bien sûr et chaque vendredi après les cours, nous établissons les objectifs de la semaine et les sessions de travail qui auront lieu selon notre emploi du temps.

De même au niveau des fichiers sources, mais surtout au niveau des commentaires : il fallait tout structurer correctement pour pouvoir comprendre et surtout ne pas oublier le pourquoi du comment nous avions fait ça. Bien entendu, nous avons grandement consolidé nos connaissances dans la structuration du code et acquis des connaissances en C++.

Pistes d'améliorations

En termes d'améliorations, nous aurions pu faire une meilleure mise en place des structures, améliorer les fonctions nécessaires au bon fonctionnement du programme.
Nous avons aussi pensé que nos commentaires n'étaient pas assez détaillé en voulant les faire trop simples à comprendre, leur placement n'est pas aussi optimal.

Le Graphe de dépendance



LE CODE :

```
#include "demineur.h"

/*
* @brief Initialise le nombre de ligne, colonnes, mine et les positions des cases minées
* @param [in] numéro de l'opération "1", lignes, colonnes, nombre de mines et les positions des cases minées
* @param [in/out] nombre de ligne, colonne, mine et les positions des cases minés
*/


void InitPB(unsigned int ligne, unsigned int colonne, unsigned int mine) {

   srand(time(NULL)); // Permet de rendre aléatoire

   int* position = new int[mine];
   int* temp = new int[ligne * colonne];

   cout << ligne << " " << colonne << " " << mine; // Entrée utilisateur
   for (unsigned int i = 0; i < mine; i++) {
      unsigned int postionmine = rand() % (ligne * colonne);

      while (temp[postionmine] == -1) {
         postionmine = rand() % (ligne * colonne);
      }

      position[i] = postionmine;
      temp[postionmine] = -1;
      cout << " " << position[i];
   }
   cout << endl;

   delete[] position;
   delete[] temp;
}
```

```
/*
* @brief Fonction permettant d'initialiser le jeu du démineur avec le nombre de lignes, colonnes, le nombre de mines, leur position et l'historique de coups
* @param [in] lignes, colonnes, nombre de mines, nombre de coups
*/

void InitJ(Grille& a, unsigned int ligne, unsigned colonne, unsigned int mine) {

   a.ligne = ligne;
   a.colonne = colonne;
   a.mine = mine;
   //Stockage des données dans la structure Grille pour les fonctions suivantes

      for (unsigned int i = 0; i < mine; ++i) {  //Choix et placement des mines stockées dans le tableau positionMines
      cin >> a.pos_Mine[i];

   }
   cin >> a.coup;
   assert(a.coup < (ligne* colonne) - a.mine);

   for (unsigned int i = 0; i < a.coup; ++i) {  //Historique de coups
      cin >> a.HCoup[i];
      cin >> a.pos_Coup[i];
   }
}
/*
* @brief Fonction triant les positions des mines dans l'ordre croissant
* @param [in] Structure Grille
*/

void triM(Grille& a) {
      //Tri position mine dans l'ordre croissant
      unsigned int t1 = 0;

      for (unsigned int i = 0; i < a.mine; i++) {
         for (unsigned int j = 0; j < a.mine; j++) {
            if (a.pos_Mine[i] < a.pos_Mine[j]) {
```

```
            t1 = a.pos_Mine[i];
            a.pos_Mine[i] = a.pos_Mine[j];
            a.pos_Mine[j] = t1;

          }
        }
      }
    }

/*
* @brief Fonction triant les positions des mines dans l'ordre croissant dans l'historique de coup.
* @param [in] Structure Grille
*/

void triH(Grille& a) {
  //Tri  mines dans l'ordre croissant dans l'historique

  unsigned int t2 = 0, tl = 0;

  for (unsigned int i = 0; i < a.coup; ++i) {
    for (unsigned int j = 0; j < a.coup; ++j) {
      if (a.pos_Coup[i] < a.pos_Coup[j]) {

        //Permet de Triez les nombres
        t2 = a.pos_Coup[i];
        a.pos_Coup[i] = a.pos_Coup[j];
        a.pos_Coup[j] = t2;

        //Permet de Triez les caractères en fonction des nombres

        tl = a.HCoup[i];
        a.HCoup[i] = a.HCoup[j];
        a.HCoup[j] = tl;

      }
    }
  }
```

```
}

/*
* @brief Fonction permettant la création de la grille
* @param [in] Grille& gr, Item** g
*/

void InitGr(Item*** a, Grille& gr) {

   // Initialise la grille

   *a = new Item * [gr.ligne];

   for (unsigned int i = 0; i < gr.colonne; ++i) {

      (*a)[i] = new Item[gr.colonne];
   }

   for (unsigned int l = 0; l < gr.ligne; l++) {

      for (unsigned int c = 0; c < gr.colonne; c++) {

         (*a)[l][c] = '.';
      }
   }
}

/*
* @brief Fonction permettant de poser les mines aléatoires de la fonction "InitPB" dans la grille
* @param [in] Structure Grille, Structure Item
*/

void PoseMine(Grille& gr, Item** a) {
   unsigned int p = 0;
   unsigned int cpt = 0;

   for (unsigned int l = 0; l < gr.ligne; ++l) {
      for (unsigned int c = 0; c < gr.colonne; ++c) {
```

```
            if (gr.pos_Mine[cpt] == p) {
                a[l][c] = M;
                cpt++;
                p++;
            }
            else {
                p++;
            }
        }
    }
}

/*
* @brief Fonction permettant de marquer ou démasquer une case
* @param [in] Structure Grille, Strucutre Item
*/

void MarqueOuDemasque(Grille& gr, Item** a) {
    int cpt = 0;
    int p = 0;
    for (unsigned int l = 0; l < gr.ligne; l++) {
        for (unsigned int c = 0; c < gr.colonne; c++) {
            if (gr.HCoup[cpt] == Marquer && gr.pos_Coup[cpt] == p) {
                a[l][c] = Marque;
                p++;
                cpt++;
            }
            else if (gr.HCoup[cpt] == Demasquer && gr.HCoup[cpt] == p) {
                if (a[l][c] == Cacher) {
                    a[l][c] = Vide;
                    p++;
                    cpt++;
                }
            }
            else {
                a[l][c] = Cacher;
                p++;
            }
```

```
      }
    }
}

/*
* @brief Affiche sur une case le nombre de mines placées sur ses cases adjacentes
* @param [in] Structure Grille, Structure Item
*/

void numerocase(Grille & gr, Item * *a) { //Mines autour

  for (unsigned int l = 0; l < gr.ligne; ++l) {      //Incrémentation des cases qui ont une mine autour d'elle. On ajoute un + aux cases adjacentes.

        for (unsigned int c = 0; c < gr.colonne; ++c) {
           if (a[l][c] == M) {
              a[l][c] = M;
              if (l >= 1 && l < gr.ligne && c >= 1 && c < gr.colonne) {
                 if (a[l - 1][c + 1] != M) {
                    a[l - 1][c + 1] += 1;
                 }
                 if (a[l - 1][c - 1] != M) {
                    a[l - 1][c - 1] += 1;
                 }
                 if (a[l - 1][c] != M) {
                    a[l - 1][c] += 1;
                 }
                 if (a[l][c - 1] != M) {
                    a[l][c - 1] += 1;
                 }
                 if (a[l][c + 1] != M) {
                    a[l][c + 1] += 1;
                 }
                 if (a[l + 1][c - 1] != M) {
                    a[l + 1][c - 1] += 1;
                 }
                 if (a[l + 1][c] != M) {
                    a[l + 1][c] += 1;
```

```
        }
        if (a[l + 1][c + 1] != M)
            a[l + 1][c + 1] += 1;

    }
}

if (l == 0 && c == 0) {
    if (a[l + 1][c + 1] != M) {
        a[l + 1][c + 1] += 1;
    }
    if (a[l][c + 1] != M) {
        a[l][c + 1] += 1;
    }
    if (a[l + 1][c] != M) {
        a[l + 1][c] += 1;
    }
}

if (l == 0 && c > 0 && c < gr.colonne) {
    if (a[l + 1][c + 1] != M) {
        a[l + 1][c + 1] += 1;
    }
    if (a[l][c + 1] != M) {
        a[l][c + 1] += 1;
    }
    if (a[l + 1][c] != M) {
        a[l + 1][c] += 1;
    }
    if (a[l + 1][c - 1] != M) {
        a[l + 1][c - 1] += 1;
    }
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;
    }
}

if (l == 0 && c == gr.colonne) {
```

```
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;
    }
    if (a[l + 1][c] != M) {
        a[l + 1][c] += 1;
    }
    if (a[l - 1][c + 1] != M) {
        a[l][c + 1] += 1;
    }
}

if (l > 0 && l < gr.ligne && c == 0) {
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
    if (a[l - 1][c + 1] != M) {
        a[l - 1][c + 1] += 1;
    }
    if (a[l][c + 1] != M) {
        a[l][c + 1] += 1;
    }
    if (a[l + 1][c + 1] != M) {
        a[l + 1][c + 1] += 1;
    }
    if (a[l + 1][c] != M) {
        a[l + 1][c] += 1;
    }
}

if (l == gr.ligne && c == 0) {
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
    if (a[l - 1][c + 1] != M) {
        a[l - 1][c + 1] += 1;
    }
    if (a[l][c + 1] != M) {
        a[l][c + 1] += 1;
```

```
    }
}

if (l == gr.ligne && c > 0 && c < gr.colonne) {
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;
    }
    if (a[l - 1][c - 1] != M) {
        a[l - 1][c - 1] += 1;
    }
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
    if (a[l - 1][c + 1] != M) {
        a[l - 1][c + 1] += 1;
    }
    if (a[l + 1][c + 1] != M) {
        a[l + 1][c + 1] += 1;
    }
}

if (l == gr.ligne && c == gr.colonne) {
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;
    }
    if (a[l - 1][c - 1] != M) {
        a[l - 1][c - 1] += 1;
    }
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
}

if (c == gr.colonne && l > 0 && l < gr.ligne) {
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
    if (a[l - 1][c - 1] != M) {
```

```
            a[l - 1][c - 1] += 1;
          }
          if (a[l][c - 1] != M) {
            a[l][c - 1] += 1;
          }
          if (a[l + 1][c - 1] != M) {
            a[l + 1][c - 1] += 1;
          }
          if (a[l + 1][c] != M) {
            a[l + 1][c] += 1;
          }
        }

      }
    }
  }
```

```
for (unsigned int l = 0; l < gr.ligne; ++l) {      //Incrémentation des cases qui ont une mine autour d'elle. On ajoute un + aux cases adjacentes.
    for (unsigned int c = 0; c < gr.colonne; ++c) {
        if (a[l][c] == M) {
            a[l][c] = M;
         if (l >= 1 && l < gr.ligne && c >= 1 && c < gr.colonne) {
            if (a[l - 1][c + 1] != M) {
                a[l - 1][c + 1] += 1;
          }
            if (a[l - 1][c - 1] != M) {
                a[l - 1][c - 1] += 1;
           }
             if (a[l - 1][c] != M) {
                 a[l - 1][c] += 1;
            }
            if (a[l][c - 1] != M) {
                a[l][c - 1] += 1;
          }
             if (a[l][c + 1] != M) {
               a[l][c + 1] += 1;
            }
             if (a[l + 1][c - 1] != M) {
                a[l + 1][c - 1] += 1;
```

```
        }
      if (a[l + 1][c] != M) {
         a[l + 1][c] += 1;
      }
       if (a[l + 1][c + 1] != M)
        a[l + 1][c + 1] += 1;

       }
    }
```

```
            if (l == 0 && c == 0) {
                if (a[l + 1][c + 1] != M) {
                    a[l + 1][c + 1] += 1;
                }
            }
        }
```

```
    /* @brief Fonction permettant d'afficher la grille
     * @param [in/out] structure grille, structure item
* @param [out] la grille*/
```

```
void AfficherGr(Grille& gr, Item** a) {

    cout << gr.ligne << " " << gr.colonne << endl;

    for (unsigned int i = 1; i <= gr.colonne; ++i) {
        cout << " ---";
    }
    cout << endl;

    for (unsigned int l = 0; l < gr.ligne; ++l) {
        for (unsigned int c = 0; c < gr.colonne; ++c) {
            cout << "| " << a[l][c] << " ";
        }
        cout << "|" << endl;
        for (unsigned int i = 1; i <= gr.colonne; ++i) {
            cout << " ---";
        }
        cout << endl;
    }
}

/*
* @brief Fonction permettant de déterminer si la partie est gagnée ou pas
* @param [in] Structure grille
* @return "GAME WON" ou "GAME NOT WON"
*/

void gagner(Grille& gr) {

    unsigned int resultat = 0;

    if (gr.coup != (gr.ligne * gr.colonne) - gr.mine) {
        resultat = 0; //game not won
    }
    else {
        for (unsigned int i = 0; i < gr.mine; ++i) {
            for (unsigned int j = 0; j < gr.mine; ++j) {
```

```
            if (gr.HCoup[i] == Demasquer && gr.pos_Coup[i] != gr.pos_Mine[j]) {
               resultat = 1; //game won
            }
            else {
               resultat = 0; //game not won
            }}}}

      if (resultat == 1) {
         cout << "GAME WON" << endl;
      }
      else {
         cout << "GAME NOT WON" << endl;
      }
   }


/*
* @brief Fonction permettant de déterminer si une partie est perdue ou pas
* @param [in] Structure grille
* return "GAME LOST" ou "GAME NOT LOST"
*/
   void perdu(Grille& gr) {

      unsigned int resultat = 0;

      for (unsigned int i = 0; i < gr.mine; i++) {
         if (gr.HCoup[gr.coup - 1] == Marquer && gr.pos_Coup[gr.coup - 1] != gr.pos_Mine[i]) {
            resultat = 0; //game lost
         }
         if (gr.HCoup[gr.coup - 1] == Demasquer && gr.pos_Coup[gr.coup - 1] == gr.pos_Mine[i]) {
            resultat = 0; //game lost
         }
         if (gr.HCoup[gr.coup - 1] == Demasquer && gr.pos_Coup[gr.coup - 1] != gr.pos_Mine[i]) {
            resultat = 1; //game not lost
         }
         if (gr.HCoup[gr.coup - 1] == Marquer && gr.pos_Coup[gr.coup - 1] == gr.pos_Mine[i]) {
            resultat = 1; //game not lost
         }
```

```
    }

    if (resultat == 0) {
        cout << "GAME LOST" << endl;
    }
    else {
        cout << "GAME NOT LOST" << endl;
    }
}


int main() {

    int cdo;
    unsigned int l , c, m;
    cin >> cdo;
    assert(cdo >= 1);
    assert(cdo <= 5);

    Grille grille;
    Item** g;
    cin >> l >> c >> m; //Entrez utilisateur

    if (cdo == 1) {
        InitPB(l, c, m);
    }

    else if (cdo == 2) {

        InitJ(grille, l, c, m);
        triM(grille);
        triH(grille);
        InitGr(&g, grille);
        PoseMine(grille, g);
        MarqueOuDemasque(grille, g);
        numerocase(grille, g);
        AfficherGr(grille, g);
    }
```

```
    else if (cdo == 3) {

        InitJ(grille, l, c, m);
        gagner(grille);

    }
    else if (cdo == 4) {

        InitJ(grille, l, c, m);
        perdu(grille);

    }
    else if (cdo == 5) {

      cout <<"nous n'avons pas réussi le sprint 5 !" <<endl;

    }


}
```

Annexe

Les fonctions en annexe sont des fonctions qui nous ont aidé à faire le projet, mais n'ont pas été explicitement demandées dans le document de cours de ce projet.

Fonction InitPB

Permet d'initialiser un problème

```
void InitPB(unsigned int ligne, unsigned int colonne, unsigned int mine) {


   srand(time(NULL)); // Permet de rendre aléatoire

   int* position = new int[mine];
   int* temp = new int[ligne * colonne];

   cout << ligne << " " << colonne << " " << mine; // Entrée utilisateur

   for (unsigned int i = 0; i < mine; i++) {
      unsigned int postionmine = rand() % (ligne * colonne);

      while (temp[postionmine] == -1) {
         postionmine = rand() % (ligne * colonne);
      }

      position[i] = postionmine;
      temp[postionmine] = -1;
      cout << " " << position[i];
   }
   cout << endl;

   delete[] position;
   delete[] temp;

}
```

Fonction InitJ

Permet d'initialiser le jeu

```
void InitJ(Grille& a, unsigned int ligne, unsigned colonne, unsigned int mine) {

   a.ligne = ligne;
   a.colonne = colonne;
   a.mine = mine;
   //Stockage des données dans la structure Grille pour les fonctions suivantes

      for (unsigned int i = 0; i < mine; ++i) {  //Choix et placement des mines stockées dans le tableau positionMines
      cin >> a.pos_Mine[i];

   }
   cin >> a.coup;
   assert(a.coup < (ligne* colonne) - a.mine);

   for (unsigned int i = 0; i < a.coup; ++i) {  //Historique de coups
      cin >> a.HCoup[i];
      cin >> a.pos_Coup[i];
   }
}
```

Fonction triM et triH
Permet de trier la position des mines ainsi que les mines en elle-même.

```
void triM(Grille& a) {
      //Tri position mine dans l'ordre croissant
      unsigned int t1 = 0;

      for (unsigned int i = 0; i < a.mine; i++) {
         for (unsigned int j = 0; j < a.mine; j++) {
```

```
void triH(Grille& a) {
   //Tri  mines dans l'ordre croissant dans l'historique
   unsigned int t2 = 0, tl = 0;
   for (unsigned int i = 0; i < a.coup; ++i) {
      for (unsigned int j = 0; j < a.coup; ++j) {
         if (a.pos_Coup[i] < a.pos_Coup[j]) {

            //Permet de Triez les nombres
            t2 = a.pos_Coup[i];
            a.pos_Coup[i] = a.pos_Coup[j];
            a.pos_Coup[j] = t2;

            //Permet de Triez les caractères en fonction des nombres
```

```
            if (a.pos_Mine[i] < a.pos_Mine[j]) {

               t1 = a.pos_Mine[i];
               a.pos_Mine[i] = a.pos_Mine[j];
               a.pos_Mine[j] = t1;}}}}
```

Fonction InitGr

Permet la création d'une grille

```
void InitGr(Item*** a, Grille& gr) {

   // Initialise la grille

   *a = new Item * [gr.ligne];

   for (unsigned int i = 0; i < gr.colonne; ++i) {

      (*a)[i] = new Item[gr.colonne];
   }

   for (unsigned int l = 0; l < gr.ligne; l++) {

      for (unsigned int c = 0; c < gr.colonne; c++) {

         (*a)[l][c] = '.';
      }}}
```

Fonction PoseMine

Pose les mines de façon aléatoire dans la grille créer au préalable

```
void PoseMine(Grille& gr, Item** a) {
   unsigned int p = 0;
   unsigned int cpt = 0;

   for (unsigned int l = 0; l < gr.ligne; ++l) {
      for (unsigned int c = 0; c < gr.colonne; ++c) {
         if (gr.pos_Mine[cpt] == p) {
            a[l][c] = M;
            cpt++;
```

```
            p++;
        }
        else {
            p++;}}}}
```

Fonction MarqueOuDemasque
Permet de marque ou demasquer une case

```
void MarqueOuDemasque(Grille& gr, Item** a) {
    int cpt = 0;
    int p = 0;
    for (unsigned int l = 0; l < gr.ligne; l++) {
        for (unsigned int c = 0; c < gr.colonne; c++) {
            if (gr.HCoup[cpt] == Marquer && gr.pos_Coup[cpt] == p) {
                a[l][c] = Marque;
                p++;
                cpt++;
            }
            else if (gr.HCoup[cpt] == Demasquer && gr.HCoup[cpt] == p) {
                if (a[l][c] == Cacher) {
                    a[l][c] = Vide;
                    p++;
                    cpt++;
                }
            }
            else {
                a[l][c] = Cacher;
                p++;
            }}}}
```

Fonction numerocase
Cette énorme fonction permet d'afficher le nombre de mines sur ses cases adjacentes

```
void numerocase(Grille & gr, Item * *a) { //Mines autour

  for (unsigned int l = 0; l < gr.ligne; ++l) {      //Incrémentation des cases qui ont une mine autour d'elle. On ajoute un + aux cases adjacentes.

        for (unsigned int c = 0; c < gr.colonne; ++c) {
```

```
if (a[l][c] == M) {
   a[l][c] = M;
   if (l >= 1 && l < gr.ligne && c >= 1 && c < gr.colonne) {
      if (a[l - 1][c + 1] != M) {
         a[l - 1][c + 1] += 1;
      }
      if (a[l - 1][c - 1] != M) {
         a[l - 1][c - 1] += 1;
      }
      if (a[l - 1][c] != M) {
         a[l - 1][c] += 1;
      }
      if (a[l][c - 1] != M) {
         a[l][c - 1] += 1;
      }
      if (a[l][c + 1] != M) {
         a[l][c + 1] += 1;
      }
      if (a[l + 1][c - 1] != M) {
         a[l + 1][c - 1] += 1;
      }
      if (a[l + 1][c] != M) {
         a[l + 1][c] += 1;
      }
      if (a[l + 1][c + 1] != M)
         a[l + 1][c + 1] += 1;

   }
}

if (l == 0 && c == 0) {
   if (a[l + 1][c + 1] != M) {
      a[l + 1][c + 1] += 1;
   }
   if (a[l][c + 1] != M) {
      a[l][c + 1] += 1;
   }
   if (a[l + 1][c] != M) {
```

```
        a[l + 1][c] += 1;
    }
}

if (l == 0 && c > 0 && c < gr.colonne) {
    if (a[l + 1][c + 1] != M) {
        a[l + 1][c + 1] += 1;
    }
    if (a[l][c + 1] != M) {
        a[l][c + 1] += 1;
    }
    if (a[l + 1][c] != M) {
        a[l + 1][c] += 1;
    }
    if (a[l + 1][c - 1] != M) {
        a[l + 1][c - 1] += 1;
    }
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;
    }
}

if (l == 0 && c == gr.colonne) {
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;
    }
    if (a[l + 1][c] != M) {
        a[l + 1][c] += 1;
    }
    if (a[l - 1][c + 1] != M) {
        a[l][c + 1] += 1;
    }
}

if (l > 0 && l < gr.ligne && c == 0) {
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
```

```
    if (a[l - 1][c + 1] != M) {
        a[l - 1][c + 1] += 1;
    }
    if (a[l][c + 1] != M) {
        a[l][c + 1] += 1;
    }
    if (a[l + 1][c + 1] != M) {
        a[l + 1][c + 1] += 1;
    }
    if (a[l + 1][c] != M) {
        a[l + 1][c] += 1;
    }
}

if (l == gr.ligne && c == 0) {
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
    if (a[l - 1][c + 1] != M) {
        a[l - 1][c + 1] += 1;
    }
    if (a[l][c + 1] != M) {
        a[l][c + 1] += 1;
    }
}

if (l == gr.ligne && c > 0 && c < gr.colonne) {
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;}
    if (a[l - 1][c - 1] != M) {
        a[l - 1][c - 1] += 1;
    }
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
    if (a[l - 1][c + 1] != M) {
        a[l - 1][c + 1] += 1;
    }
```

```
    if (a[l + 1][c + 1] != M) {
        a[l + 1][c + 1] += 1;
    }
}

if (l == gr.ligne && c == gr.colonne) {
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;
    }
    if (a[l - 1][c - 1] != M) {
        a[l - 1][c - 1] += 1;
    }
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
}

if (c == gr.colonne && l > 0 && l < gr.ligne) {
    if (a[l - 1][c] != M) {
        a[l - 1][c] += 1;
    }
    if (a[l - 1][c - 1] != M) {
        a[l - 1][c - 1] += 1;
    }
    if (a[l][c - 1] != M) {
        a[l][c - 1] += 1;
    }
    if (a[l + 1][c - 1] != M) {
        a[l + 1][c - 1] += 1;
    }
    if (a[l + 1][c] != M) {
        a[l + 1][c] += 1;}}}}}
```

Fonction AfficherGr
Permet d'afficher la grille

```
void AfficherGr(Grille& gr, Item** a) {
```

```
    cout << gr.ligne << " " << gr.colonne << endl;

    for (unsigned int i = 1; i <= gr.colonne; ++i) {
       cout << " ---";
    }
    cout << endl;

    for (unsigned int l = 0; l < gr.ligne; ++l) {
       for (unsigned int c = 0; c < gr.colonne; ++c) {
          cout << "| " << a[l][c] << " ";
       }
       cout << "|" << endl;
       for (unsigned int i = 1; i <= gr.colonne; ++i) {
          cout << " ---";
       }
       cout << endl;
    }
}
```

Fonction gagner et perdu

Permet de déterminer si une partie est gagné ou perdue

```
void gagner(Grille& gr) {
```

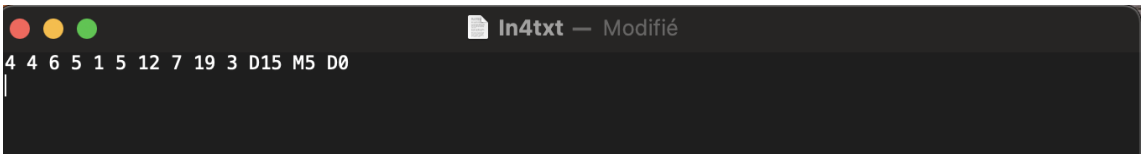
```
void perdu(Grille& gr) {

    unsigned int resultat = 0;

    for (unsigned int i = 0; i < gr.mine; i++) {
       if (gr.HCoup[gr.coup - 1] == Marquer && gr.pos_Coup[gr.coup - 1] != gr.pos_Mine[i]) {
```

```
    unsigned int resultat = 0;

    if (gr.coup != (gr.ligne * gr.colonne) - gr.mine) {
      resultat = 0; //game not won
    }
    else {
      for (unsigned int i = 0; i < gr.mine; ++i) {
        for (unsigned int j = 0; j < gr.mine; ++j) {
          if (gr.HCoup[i] == Demasquer && gr.pos_Coup[i] !=
gr.pos_Mine[j]) {
            resultat = 1; //game won
          }
          else {
            resultat = 0; //game not won
          }
        }
      }
    }

    if (resultat == 1) {
      cout << "GAME WON" << endl;
    }
    else {
      cout << "GAME NOT WON" << endl;
    }
  }
```

Jeux d'essai

Le sprint le plus haut que nous avons réussi à atteindre et le sprint numéro 4

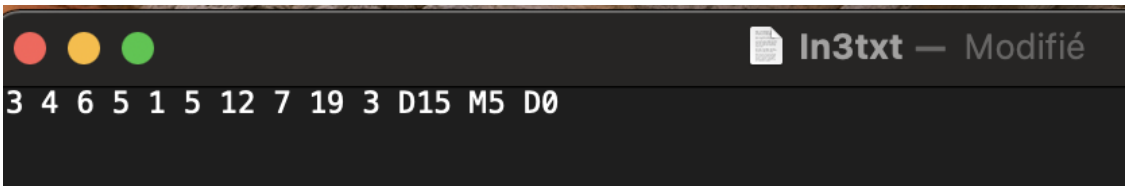
fichier in pour le sprint 4



fichier out pour le sprint 4

```
> ./main
4 4 6 5 1 5 12 7 19 3 D15 M5 D0
GAME NOT LOST
>
```

fichier in pour le sprint 3

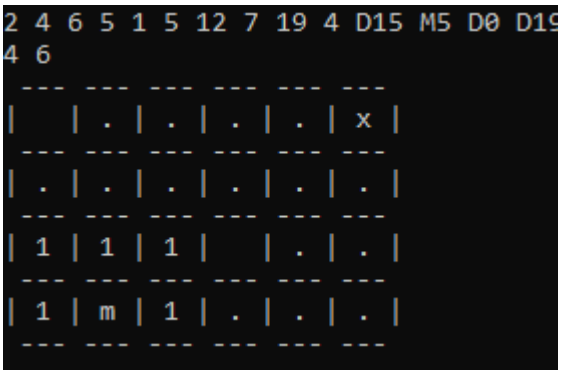


fichier out pour le sprint 3

```
> ./main
3 4 6 5 1 5 12 7 19 3 D15 M5 D0
GAME NOT WON
>
```

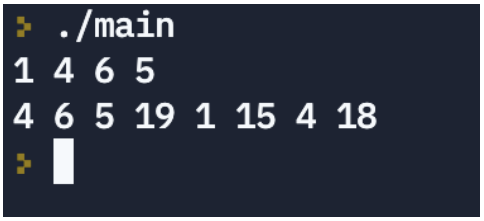
fichier in pour le sprint 2

fichier out pour le sprint 2



fichier in pour le sprint 1

fichier out pour le sprint 1



MERCI D'AVOIR LU NOTRE RAPPORT !

Yanis Segdauh, Elias Ait-Khelifa