Rapport de projet de SDA

Nils MITTELHOCKAMP groupe 112
Louis MASSON groupe 103

Production d’un démineur

TABLE DES MATIÈRES

INTRODUCTION AU PROJET

Le but de ce projet est de concevoir un programme permettant de jouer au démineur à travers des commandes. Ce programme doit être capable de :

- Créer un problème, c'est-à-dire générer des bombes aléatoirement sur un plateau à l'aide de dimensions données
- Créer une grille, c'est-à-dire suivre l'avancement d'une partie à l'aide d'un problème et d'un historique de coups et afficher la grille correspondante.
- Vérifier si la partie est gagnée
- Vérifier si la partie est partie
- Lire une grille et indiquer un coup possible

Les objectifs de ce projet sont donc de produire un code optimisé répondant à ces commandes à l'aide des notions vues en cours ; l'allocation dynamique, les flux…

INTRODUCTION AU PROJET

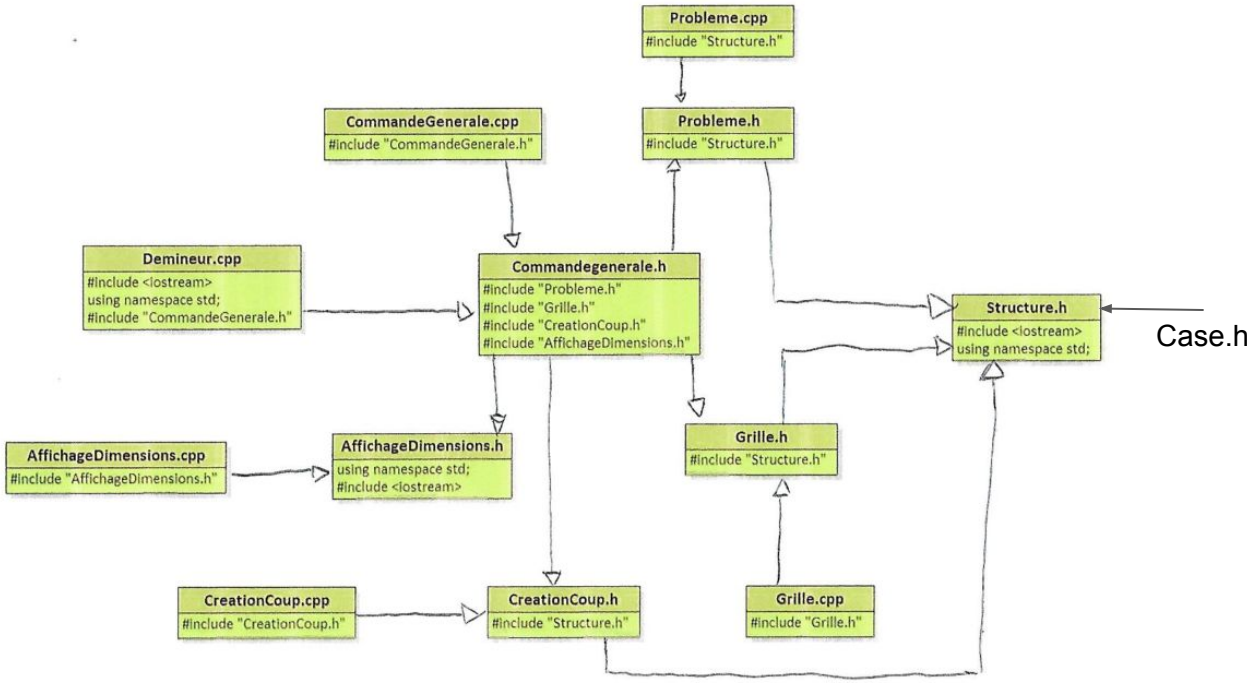
Afin de créer le démineur, nous avons décidé de créer des types énumérés afin de représenter les types de cases du plateau et les différents états d'une partie de jeu. Pour représenter les différentes cases du plateau, nous avons créé une structure *Case* composée de différentes variables nous permettant de représenter toutes les différentes cases de jeu possibles.

```
//Strcuture d'une case
struct Case
{
    bool decouverte = false; //Case découverte ?
    bool marquee = false; //Case marquée ?
    typeCase c = RIEN; //Type de la case
    unsigned int nbMinesVoisines = 0; //Nombre de mines avoisinantes
};
```

```
//Type d'une case: vide, mine voisine ou minée
enum typeCase { RIEN = 0, DANGER = 1, MINE = 2 };

//Type de partie: perdue, gagnee ou en cours
enum Partie { PERDUE = 0, GAGNEE = 1, EN_COURS = 2 };
```

Graphe de dépendances



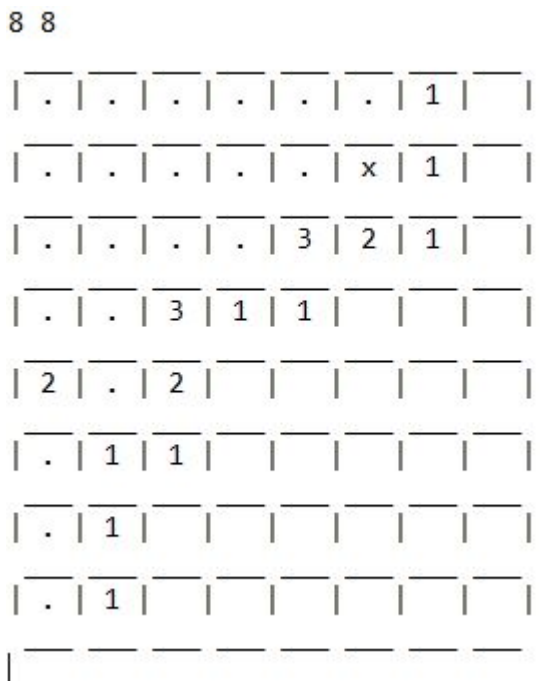
JEUX D’ESSAIS

IN 1:

- 1 8 8 0 → 8 8 0
- 1 2 2 2 → 2 2 2 1 2

IN 2:

- 2 8 8 6 13 25 12 56 19 33 4 D15 M13 D32 D62 →



JEUX D’ESSAIS

IN 2:

- 2 8 8 6 13 25 12 56 19 33 4 D15 M25 D33 D62 →

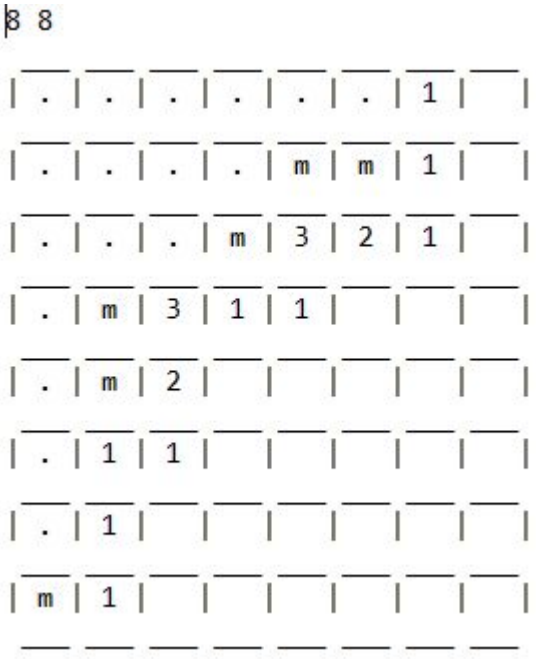
IN 3:

- 3 8 8 6 13 25 12 56 19 33 4 D15 M13 D32 D62

  → game not won

- 3 8 8 3 13 25 12 4 D15 M13 M12 M25

  → game won



JEUX D’ESSAIS

IN 4:

- 4 8 8 6 13 25 12 56 19 33 4 D15 M13 D32 D62

  → game not lost

- 4 8 8 6 13 25 12 56 19 33 4 D15 M25 D33 D62

  → game lost

JEUX D’ESSAIS

IN 5:

```
5 8 8
 __ __ __ __ __ __ __ __
|  .|  .|  .|  .|  .|  .|  1|   |
 __ __ __ __ __ __ __ __
|  .|  .|  .|  .|  .|  x|  1|   |
 __ __ __ __ __ __ __ __
|  .|  .|  .|  .|  3|  2|  1|   |
 __ __ __ __ __ __ __ __
|  .|  .|  3|  1|  1|   |   |   |
 __ __ __ __ __ __ __ __
|  2|  .|  2|   |   |   |   |   |
 __ __ __ __ __ __ __ __
|  .|  1|  1|   |   |   |   |   |
 __ __ __ __ __ __ __ __
|  .|  1|   |   |   |   |   |   |
 __ __ __ __ __ __ __ __
|  .|  1|   |   |   |   |   |   |
 __ __ __ __ __ __ __ __
```

→ D24

DIFFICULTÉS RENCONTRÉES

I. Convertir la position d'une bombe en indices

Nous avons répondu à ce problème à l'aide des formules :

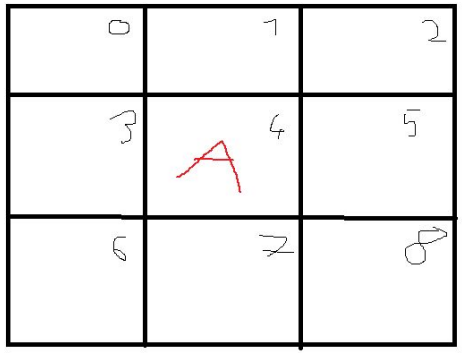
Indice ligne : ***(position - (position % nombre de colonnes)) / nombre de colonnes***

Indice colonne : ***position % nombre de colonnes***

Exemple : Plateau 3x3

Position de A :

- *(4 - (4%3))/3 = 1*
- *4 % 3 = 1*



DIFFICULTÉS RENCONTRÉES

II. Vérifier les mines voisines à une case

Afin d'indiquer si une case comporte des mines adjacentes, nous avons conçu une quadruple boucle qui parcourt chaque case du plateau, et pour chacune de ses cases, le programme parcourt les 9 cases adjacentes (si elles existent) à l'aide d'une double boucle à la recherche de mines

```
/*Quadruple boucle qui permet de vérifier les 9 cases ajacentes à une case
* On soustrait une valaure à un indice pour obtenir une case adjacente */
for (unsigned int i = 0; i < nbLignes; i++) //Indice ligne
{
    for (unsigned int j = 0; j < nbColonnes; j++) //Indice colonne
    {
        for (int k = -1; k <= 1; k++) //Valeur à soustraire à l'indice colonne pour obtenir une case voisine
        {
            for (int l = -1; l <= 1; l++) //Valeur à soustraire à l'indice ligne pour obtenir une case voisine
            {
                if ((i - l) >= 0 && (j - k) >= 0 && (i - l) < nbLignes && (j - k) < nbColonnes) //On verifie que la case existe (indices >= 0
                {
                    if (plateau[i - l][j - k].c == MINE && plateau[i][j].c != MINE) //Si une case voisine est une mine
                    {
                        plateau[i][j].c = DANGER;
                        plateau[i][j].nbMinesVoisines = plateau[i][j].nbMinesVoisines + 1;
                    }
                }
            }
        }
    }
}
```

DIFFICULTÉS RENCONTRÉES

III. Récupérer un coup

Afin de récupérer un coup joué, nous avons décidé de convertir la position entrée en entier non signé.

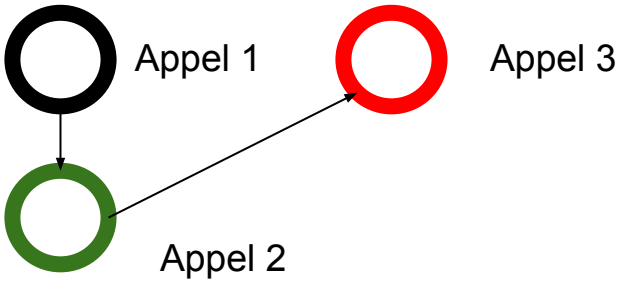
En effet, la position est égale à la somme des produits des chiffres de la position et de leurs poids. On caste un caractère en entier non-signé, que l'on multiplie par son poids (10 puissance sa position dans le coup). On additionne ce produit au résultat précédent pour obtenir la position finale.

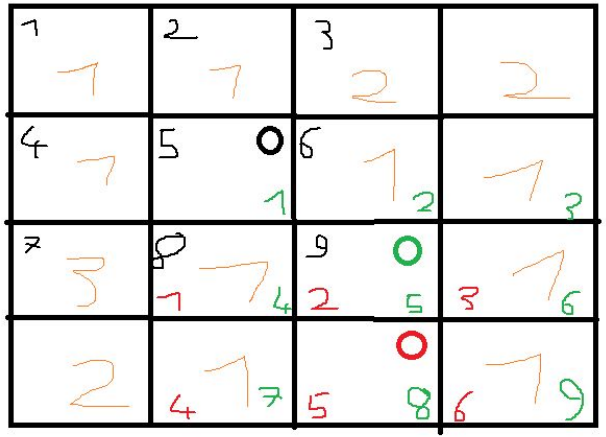
```
position = ((unsigned int)coup[j] - 48) * ((unsigned int)pow(10, (strlen(coup) - 1 - j))) + position;
```

DIFFICULTÉS RENCONTRÉES

IV. Découvrir les cases

Afin de découvrir les cases vides adjacentes lorsqu'une case vide est découverte, nous avons conçu une fonction récursive qui découvre les 9 cases adjacentes à une case vide et qui s'appelle elle même dès lors qu'elle trouve une autre case vide. Cela a pour conséquence de parcourir l'ensemble des cases vides adjacentes dès lors qu'une case vide est trouvée.





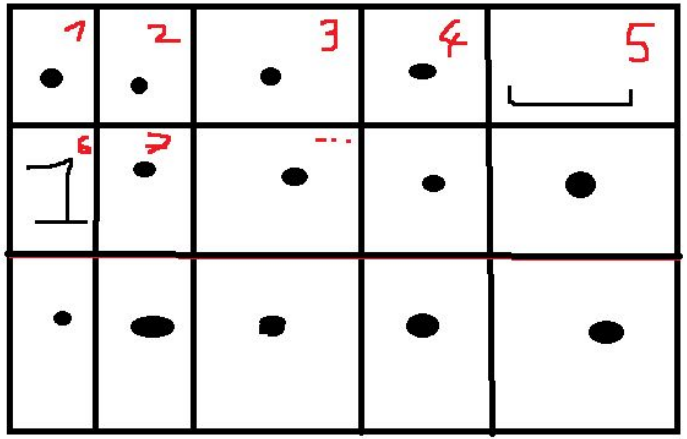
DIFFICULTÉS RENCONTRÉES

V. Récupérer une grille

Pour récupérer une grille de jeu, nous avons décidé de récupérer une grille caractère par caractère et d'adapter la position à chaque fois qu'une case est rencontrée.

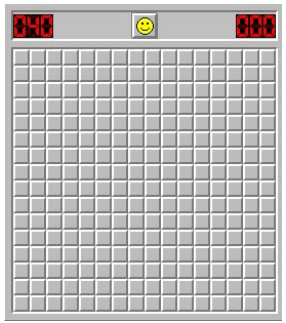
La position de chaque case est incrémentée lorsque le programme rencontre:

- Un point
- Un chiffre
- La lettre x
- Deux barres '|' qui se suivent et étant sur la même ligne, signifiant la présence d'une case vide entre les deux barres.



AMÉLIORATIONS POSSIBLE

- Afin d'optimiser notre programme, il serait possible de mettre au point un moyen de vérifier les mines voisines à une case sans parcourir l'ensemble du plateau, par exemple au moment de la création ou de l'ajout des bombes.
- Afin d'améliorer l'expérience utilisateur, il serait possible d'écrire une fonction qui vérifie la saisie de l'utilisateur à la recherche d'erreurs et qui lui indiquerait si elle venait à en trouver.
- Pour faciliter la saisie des coups, il pourrait être intéressant de développer une interface graphique afin de faciliter l'entrée des coups.



BILAN DU PROJET

Ce projet de démineur nous a permis de consolider nos connaissances sur le langage C++. En effet, ce projet nous as permis de voir les notions de :

- Tableau dynamique à deux dimensions
- Les flux, en particulier le flux cin avec la récupération de grilles
- Fonction récursive
- Manipulation de types énumérés et de tableaux de structures

De plus, ce projet nous as forcé à avoir une vrai réflexion sur la manière de répondre aux problèmes et aux objectifs posés, notamment en faisant des schémas et des essais sur papier, ou en créant des jeux d’essais personnalisés.

BILAN DU PROJET

Schémas qui nous ont aidé lors de la conception du démineur