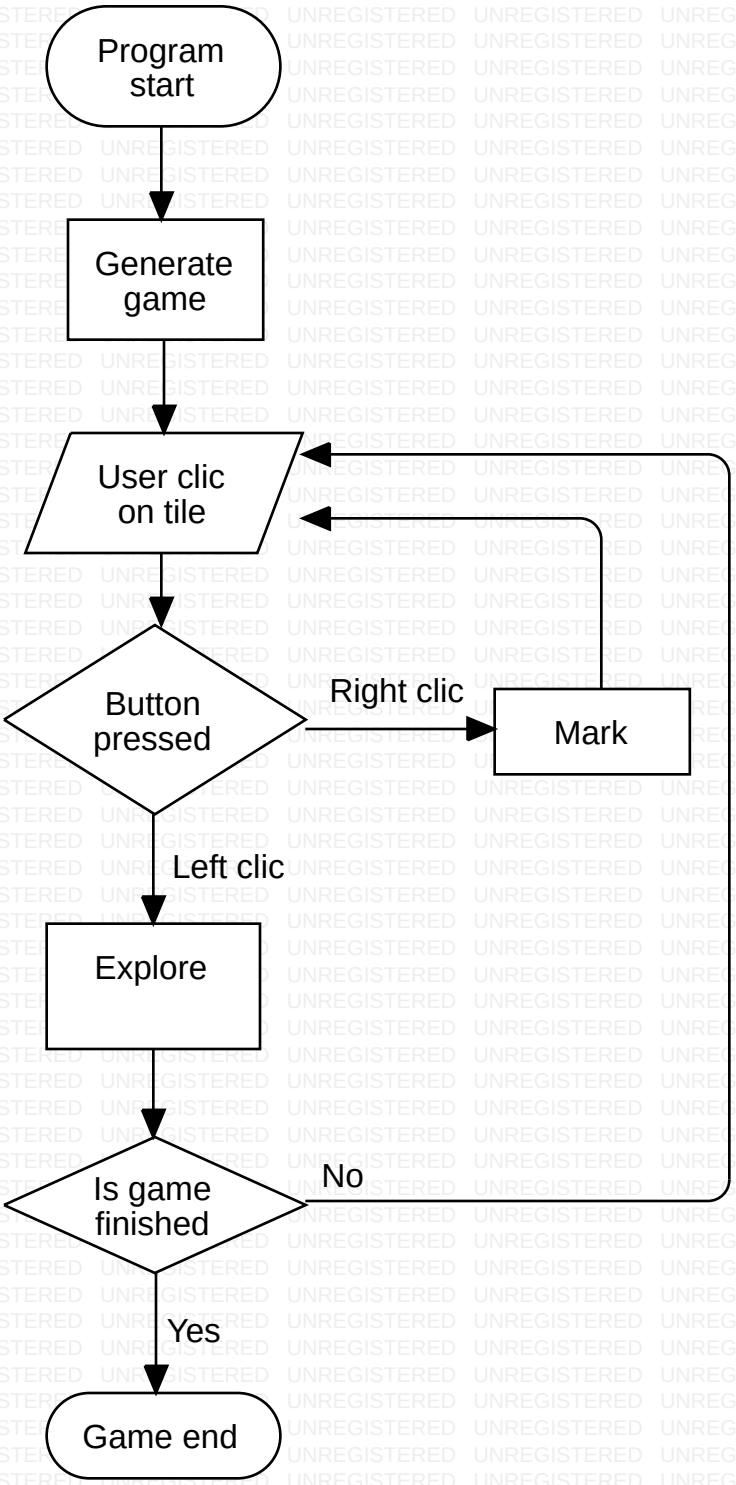
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

RAPPORT DE LABORATOIRE #1
GPA665

PAR
Maxime ROYAL

MONTRÉAL, le 12 Janvier 2020

Flowchart Flowchart Diagram



Grille d'évaluation

En guise d'autoévaluation, imprimez et remplissez vous-même cette section. Mentionnez une référence (nom de fichier et numéro de ligne) lorsque demandé pour faciliter la vérification par le correcteur.

	Spécifications techniques	**Oui**	**Non**	**Référence (fichier, ligne)**
2.5	Options offertes sur la taille de la grille	☑	☐	Minesweeper.cpp ligne 123
2.5	Différentes difficultés offertes	☑	☐	Minesweeper.cpp ligne 19
5	Affichage correct de la grille	☑	☐	_____
5	Comportement attendu du jeu	☑	☐	_____
5	Fin de la partie	☑	☐	Minefield.cpp ligne 323
	Spécifications de programmation	**Oui**	**Non**	
5	Structures de données imposées : tableau 2D, *CaseChampMine*, *NiveauDifficulte*	☑	☐	Grid.h Box.h Minefield.h
5	Gestion dynamique de la mémoire et libération de l'espace alloué	☑	☐	Grid.cpp Vector.h
5	Utilisation de fonction avec passage de paramètre par référence	☑	☐	Grid.h ligne 15
5	Utilisation d'une fonction récursive pour l'exploration des cases de la grille	☑	☐	Box.h ligne 58
(-10)	Aucune variable globale	☑	☐	

	Qualité logicielle	**T.B.**	**Bon**	**Pauvre**	**Référence (fichier, ligne)**
5	Modularité : usage généralisé de fonctions et regroupement logique en modules	☑	☐	☐	Grid.h Vector.h (Templates)
5	Robustesse : Précaution pour éviter les problèmes de données erronées, etc.	☑	☐	☐	Grid.h Vector.h (comportement wrappé en clases)
5	Documentation : Commentaires pertinents qui accélèrent la compréhension du code	☐	☑	☐	_____
5	Clarté du code : Code concis et court utilisant les fonctions adaptées	☐	☑	☐	Box.h ligne 181 seule fonction à plus de 25 lignes
60	**TOTAL**				

Section réservée au correcteur :

/ 10	Organigramme
/ 30	Réponses aux questions
/ 60	Respect des spécifications
/ 100	**Total**

Q1 : Quelle est la pertinence de la structure de données utilisée dans ce laboratoire? En quoi cette structure est-elle adaptée ou non au problème de la gestion du contenu et de l'affichage d'une grille? Donnez les avantages de la structure de données imposée pour ce problème. Parlez de modifications que vous apporteriez à la structure de données pour l'améliorer. (15)

R1 : Puisque le plateau de démineur est en effet une grille à 2 dimensions, la structure présentée reflète bien ce concept. De plus cette implémentation par tableau de tableau permet d'utiliser facilement grille[x][y] pour accéder à un élément de la structure. Finalement, un tableau de tableaux facilite les opérations d'insertions de rangés, assumant que le premier tableau représente les rangées comme dans une matrice, puisque seulement une petite partie des données doivent être déplacées versus la quasi-totalité dans une implémentation différente, tel que l'émulation de 2 dimensions avec un tableau 1D.

Cependant, cette implémentation nécessite deux opérations de déréférence pour accéder à un élément, ce qui est très coûteux. L'implémentation par émulation nous permet de réduire le nombre d'opération de déréférence à 1, et ce, peu importe le nombre de dimension requises. Puisque cette application spécifique ne requiert pas de redimensionnement ou d'insertion de la grille une fois que la partie est commencé, il serait peut-être préférable d'opter pour cette approche, spécialement si le projet est fait un C++ qui nous permettrait d'encapsuler toutes les opérations moins intuitives dans une classe et laisser l'utilisateur interagir avec la structure au travers de méthodes d'accesseurs et de mutateurs.

Veuillez prendre note que mon implémentation diffère légèrement puisque j'ai encapsulé le comportement d'un tableau dynamique dans une classe Template nommée Vecteur basée sur la classe du même nom dans la STL. Ma grille n'est donc pas un tableau dynamique de tableaux dynamiques, mais bien un vecteur de vecteurs. De ce fait, l'accesseur [x][y] ne m'est pas disponible. Pour remédier à cet inconvénient, j'ai surchargé l'opérateur () pour y passer x et y, grille(x, y).

Q2 : Soit un changement aux spécifications du jeu. Pour une difficulté additionnelle, il arrive qu'une ligne ou une colonne complète (parsemée aléatoirement de cases libres ou de mines) soit insérée à quelque part dans la grille, et ce pendant une partie en cours! L'ajustement des chiffres indiquant la quantité de mines voisines dans les cases adjacentes à cette nouvelle ligne ou colonne est effectué tout de suite après l'insertion. Proposer une façon de faire une modification à la structure de données pour faciliter l'implémentation de cette nouvelle option. (10)

R2 : Mon implémentation de la structure au travers de la classe Grid<class T> supporte déjà l'ajout d'une ou plusieurs colonnes ou rangées avec ses méthodes :

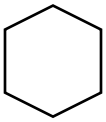
```
void insertRows(size_type row, const Grid<T>& elements);
void insertRow(size_type row, const Vector<T>& elements);
void insertCols(size_type col, const Grid<T>& elements);
void insertCol(size_type col, const Vector<T>& elements);
```

La structure de donnée en elle-même n'a pas à être fondamentalement changée pour permettre l'ajout facile de colonnes ou de rangées. De plus, ma classe Minefield, qui regroupe les

comportements du jeu de démineur, possèdes deux fonctions servant à mettre à jour les valeurs de chaque case selon le nombre de mines qui les entourent.

Q3 : Après des ventes incroyables, votre logiciel a besoin de nouveauté pour continuer à conquérir le cœur du public. Un collègue propose une idée révolutionnaire : des cases hexagonales! Proposez une méthode ou une modification à la structure de données qui permettrait la réalisation de cette nouvelle grille. (5)

R3 : Une grille hexagonale peut être naviguée avec un système d'axe à 3 composantes. Imaginons nos hexagones comme ayant leurs pointes sur l'axe vertical et présentant des plats à l'axe horizontal.



Un premier axe aurait son sens positif à 0° et négatif à 180°. Le deuxième axe serait orienté à 60°par rapport à l'axe horizontal et le troisième axe pointerait à 120°. Avec cette convention, nous pouvons décrire la position de n'importe quelle case au moyen d'une combinaison de déplacement selon au moins 2 des 3 axes. Ceci même évidement au problème qu'une case peut être référencée par plusieurs coordonnées. Cependant, avec une bonne méthodologie d'attribution et une consistance dans nos appels, nous pouvons contourner de problème. De plus, cette convention nous facilite la tâche lorsque vient le temps d'accéder aux voisins car ils sont facilement référençable avec [+1, 0, 0], [-1, 0, 0], [0, +1, 0], [0, -1, 0], [0, 0, +1], [0, 0, -1]. La façons la plus simple d'implémenter cette convention sous forme de structure de donnée serait d'avoir une structure tridimensionnelle, où chaque dimension représente un des axes.