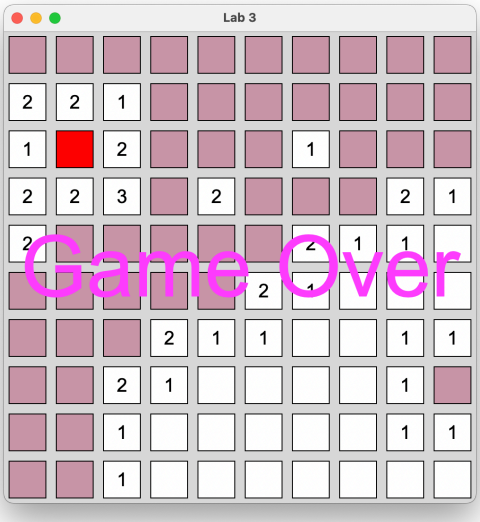
INFO-F-202
Langages de programmation 2
Université libre de Bruxelles
Laboratoire 3

Justin Dallant John Iacono
Yannick Molinghen Alexis Reynouard

28 Septembre 2021



1 Tu devrais avoir ...

Vous devriez avoir regardé les vidéos suivantes :

1. Le cours : structure, logistique et les notes
2. Pourquoi C++
3. Encapsulation
4. Durée de vie des objets I : variables automatiques
5. Durées de vie des variables
6. Pointeurs
7. Passage de paramètres
8. Constructeurs

Rappel : Les vidéos que vous devez regarder chaque semaine sont postées sur UV.

2 Objets et pointeurs

Cette semaine, nous continuerons à pratiquer la programmation orientée objet de base, et introduirons l'utilisation des pointeurs.

Le but aujourd'hui sera de coder le jeu vidéo démineur. Si vous ne connaissez pas cet jeu, allez par exemple sur minesweeper online et jouez jusqu'à ce que vous le compreniez.

De notre point de vue, le démineur est intéressant car nous pouvons le coder seulement avec les classes Canvas et Cell que nous avons utilisées la semaine dernière en ajoutant uniquement des méthodes et des variables d'instance. En plus de la classe Rectangle de la semaine dernière, je vous donne le code d'une classe appelée Text que vous pouvez utiliser pour dessiner du texte à l'écran.

3 Étapes

Je recommande de suivre ces étapes. Je répète que tout ce que vous devez faire à chaque étape est de changer Canvas et Cell. Vous devrez ajouter des variables d'instance à ces classes pour "se souvenir" de leur état, et vous voudrez ajouter des méthodes pour abstraire chacune des tâches que vous devez effectuer.

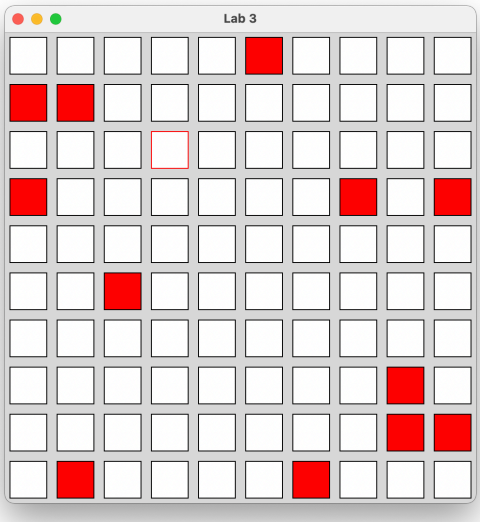
1. Comprenez le code avec lequel vous devez commencer. Commencez avec votre code de la semaine dernière ou avec lab3.cpp. Dans lab3.cpp, j'ai fait quelques ajouts par rapport à la solution de la semaine dernière :
   — Il existe une classe Text et une classe Rectangle pour dessiner du texte et des rectangles. La classe Text est nouvelle. Vous pouvez les initialiser avec le constructeur et appeler draw sur eux lorsque vous voulez dessiner. Ils ont maintenant des setters et des getters pour tous leurs paramètres. Vous ne devriez pas avoir besoin de modifier ces classes.
   — Le constructeur de Canvas stocke maintenant les cellules dans un vecteur 2D, pas 1D, c'est-à-dire un vecteur de vecteurs. Celui-ci est créé comme suit :

```
    for (int x=0;x<10;x++){
        cells.push_back({});
        for (int y=0;y<10;y++)
           cells[x].push_back({{50*x+25,50*y+25},40,40});
    }
```

   Les méthodes de Canvas comme draw ont maintenant été modifiées pour boucler sur toutes les cellules de ce vecteur de vecteurs :

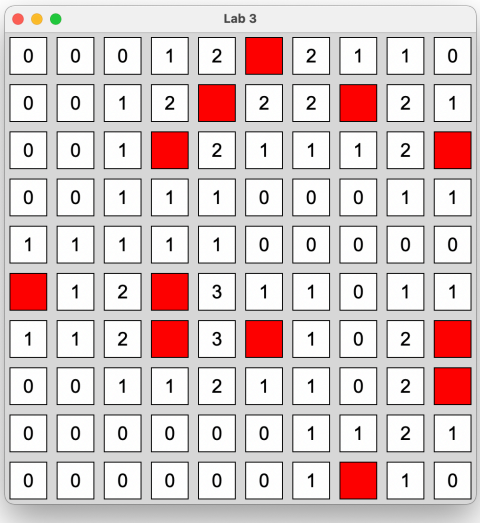
```
void Canvas::draw() {
    for (auto &v:cells)
        for (auto &c:v){
            c.draw();
        }
```

   Mais, plus important encore, vous pouvez accéder à la cellule en (3,4) à l'intérieur du canevas en écrivant `cells[3][4]`.
   — La possibilité d'activer et de désactiver les cellules a été supprimée.
   — j'ai ajouté `srand(time(0));` au `main`. Sans cela, vous obtenez les mêmes nombres aléatoires à chaque fois que vous exécutez le programme.
2. Nous commencerons par désigner $\frac{1}{8}$ des cellules comme "bombes". Vous pouvez utiliser la fonction `rand()` pour obtenir des nombres aléatoires. Dessinez les bombes en rouge et les autres cellules en blanc. La sortie devrait ressembler à :



Cette étape passe en revue des bases. Vous devez comprendre :
— Où stocker si une cellule est une bombe ou non.
— Comment initialiser si une cellule est une bombe ou non.
— Comment dessiner une cellule différemment selon qu'il s'agit ou non d'une bombe.

3. Dans cette étape, vous devez dessiner, pour chaque cellule qui n'est pas une bombe, le nombre de cellules adjacentes (y compris les diagonales) qui sont des bombes.



C'est l'étape la plus difficile de ce que vous devez faire aujourd'hui. Pour ce faire, je souhaite que vous autorisiez chaque cellule à accéder directement à ses cellules voisines. Plus précisément, je veux que chaque cellule ait une variable d'instance appelée `neighbors` (voisins) de type `vector<Cell *>`. C'est-à-dire que chaque cellule aura un vecteur de POINTEURS vers ses voisins. En général les cellules ont 8 voisines, mais celles sur les bords en auront moins.

Pour les compléter, vous devrez déterminer :
— Comment initialiser `neighbors` comme variable d'instance de chaque cellule pour avoir les pointeurs vers toutes ses cellules voisines. Astuce : cela ne peut pas être fait pendant le constructeur car les cellules voisines peuvent ne pas encore exister. Si

cela n'est pas fait dans le constructeur, cela signifie que vous devez écrire une autre méthode pour le faire que vous appelez une fois toutes les cellules ont été construites.
- Comment une cellule utilise ses voisins pour calculer le nombre de cellules voisines qui ont des bombes. Astuce : codez une méthode `NearBombCount` dans la cellule pour abstrait la logique de cette tâche.
- Bien sûr, vous devez modifier la méthode de dessin de cell pour réellement dessiner les nombres. Rappeler `to_string(i)` convertit les types intégrés en chaîne.

4. Maintenant que toutes les données de base sont là, les dernières étapes consistent à faire en sorte que votre programme se comporte comme le jeu *Démineur*. Pensez pour chacune des étapes ci-dessous quels sont les états supplémentaires et à qui il incombe de retenir ces états.
   - Au lieu d'afficher les bombes et les nombres au début, laissez toutes les cellules vides et lorsque vous cliquez sur une cellule, la bombe ou le nombre apparaît.
   - Si l'utilisateur appuie sur la barre d'espace, un nouveau jeu est créé au hasard et il commence avec tout vide.
   - Si l'utilisateur clique sur une bombe, "Game Over" s'affiche et l'utilisateur ne peut rien faire d'autre que d'appuyer sur la barre d'espace pour démarrer une nouvelle partie.
   - On n'affiche pas de zéro quand il n'y a pas de bombe dans les voisins. De plus, tout comme dans le vrai jeu, si vous cliquez sur un zéro, tous les voisins de la cellule sont découverts (car ils sont en sécurité), et si l'un des voisins est un zéro, leurs voisins sont découverts et le processus se poursuit jusqu'à ce que ça se termine. Ce n'est pas difficile à faire puisque nous avons le vecteur de pointeurs vers les voisins.
   - Si l'utilisateur a découvert toutes les cellules non explosives, affichez "vous gagnez" à l'écran.