Rapport Projet Démineur

Remise 3 (Graphique)

Youssef Snoussi (G41385)

Thales Storino Almeida (G40634)

(Groupe D212)

13/05/2018

Table de matières

Introduction

Le projet de DEV4 en deuxième bloc est important car il sera un plus pour notre avenir professionnel. De notre point de vue, ce projet présente deux caractéristiques fondamentales: d'une part le premier contact avec un développement d'une application qui contiendra un mode graphique, et d'autre part en tant que futur informaticien, élargir son champ de compétences avec le langage c++.

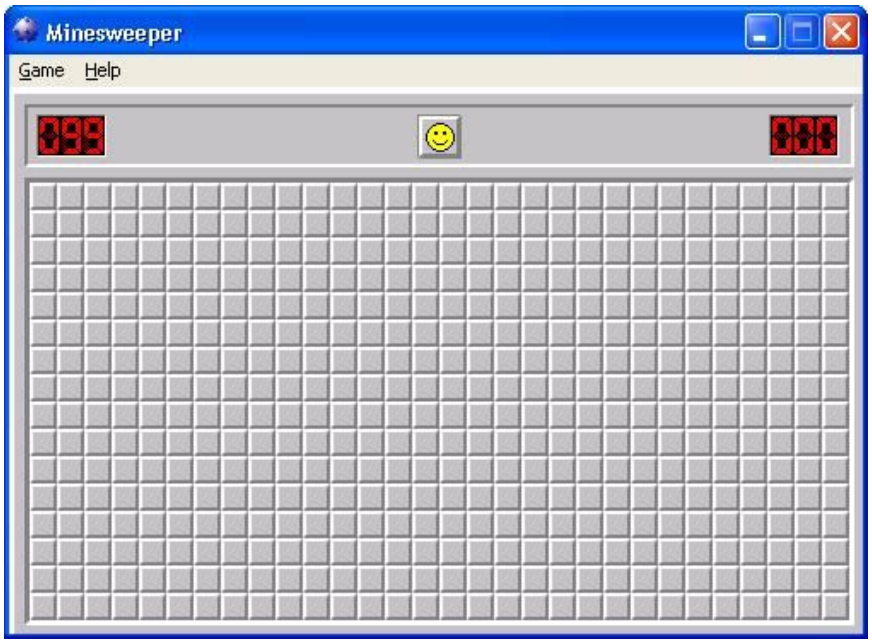
Le but de ce projet est d'implémenter une version complète du jeu « Démineur ». Pour cette deuxième remise, nous allons présenter les modifications/améliorations par rapport à la remise précédente.

De nouveau, il est possible qu'au fur et à mesure du codage par la suite, il y ait quelques changements au niveau des classes, méthodes ou variables, car il arrive souvent qu'on se rende compte pendant le codage qu'on aurait pu améliorer quelques méthodes, en les simplifiant/rendant plus efficace.

Le jeu “Démineur”

Démineur (*Minesweeper* en anglais) est un très ancien jeu vidéo qui a fait ses premières apparitions dans les années 60, et aujourd’hui il est distribué sur diverses plateformes. Une partie, disputée seule, consiste à détecter des mines sur un plateau. Ceci se joue sur un plateau rectangulaire de longueur et largeur arbitraires, dont le contenu est caché. Certaines cases contiennent des mines. À chaque tour, l’utilisateur peut soit choisir de révéler une case, soit d’en marquer une comme « minée » (en ce moment, un drapeau apparaît sur la case). S’il révèle une case minée, il a perdu la partie. Il remporte la partie quand toutes les cases ne contenant pas de mines sont révélées. Au plus vite l’utilisateur fini le jeu, mieux sera son score.

Pour le développement de ce jeu, nous allons travailler en Orienté Objet, le divisant en plusieurs classes, chaqu’une avec un rôle spécifique, mais qui sont toutes liées entre elles.



Démineur sous Windows XP

Le diagramme de classes

Diagramme de la première remise:

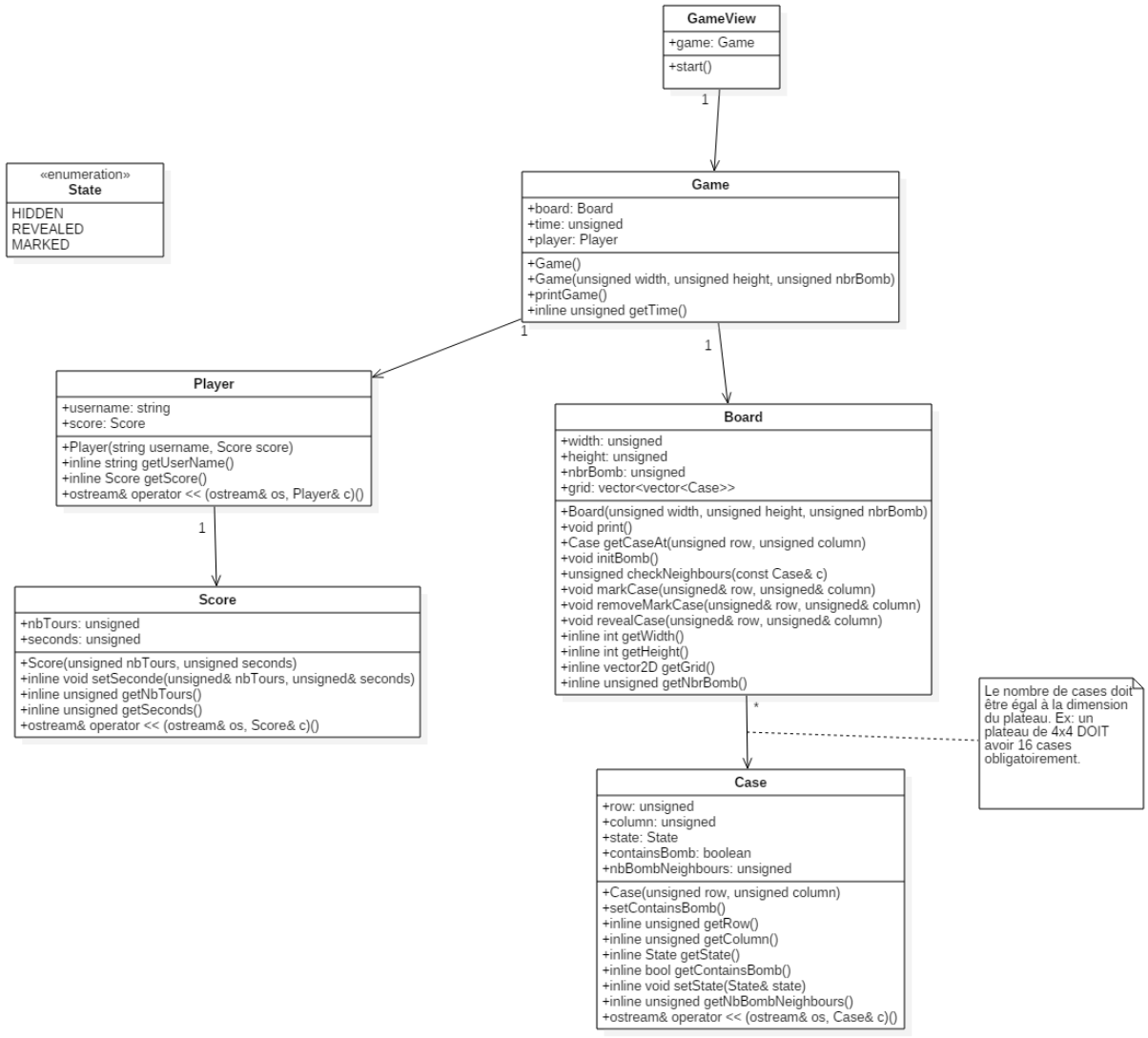
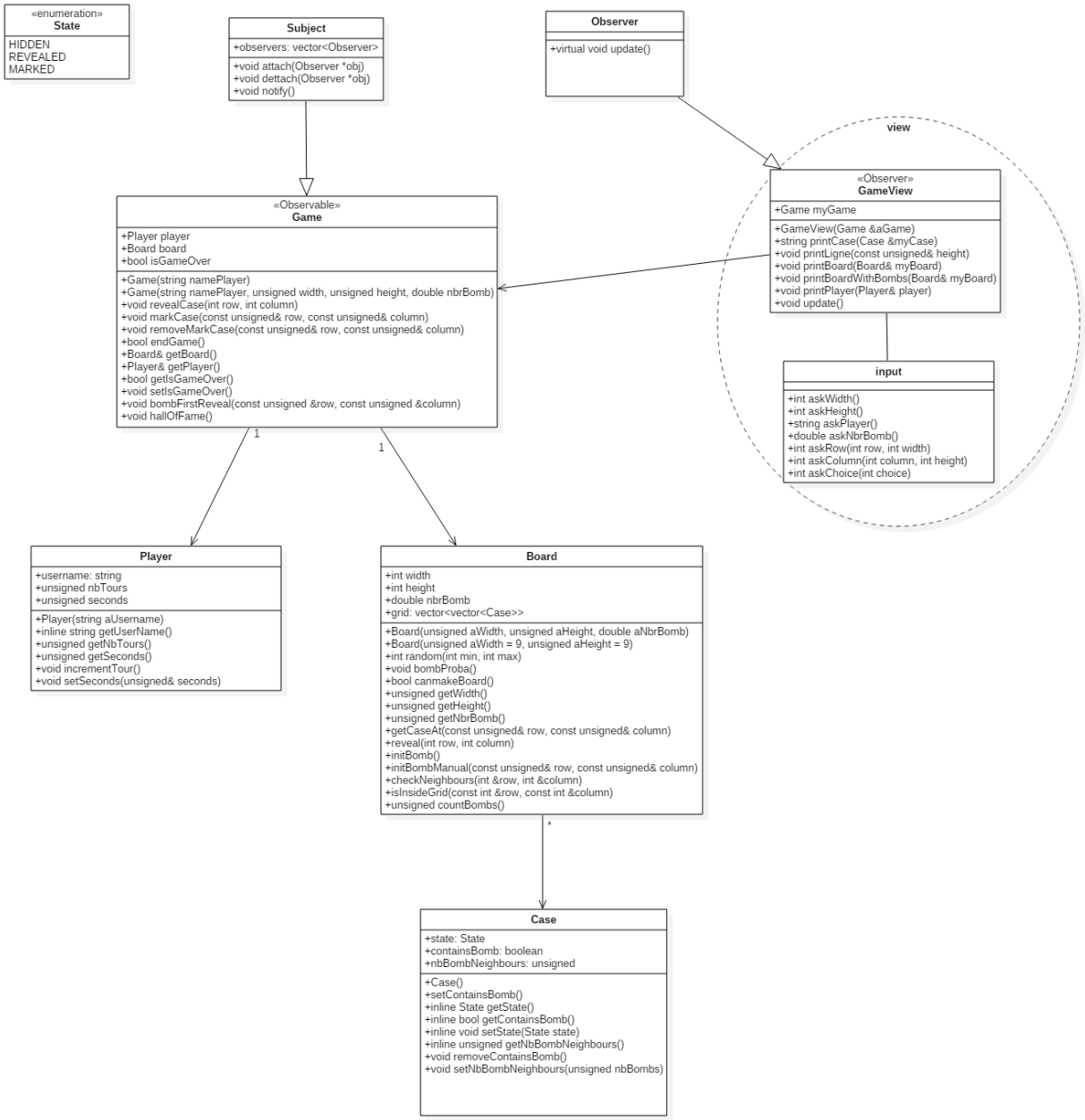


Diagramme de la deuxième remise:



Les Classes (Première remise)

Board

C'est la classe qui représente le plateau du jeux. Le plateau pourra être construit avec une taille par défaut, ou avec une taille personnalisée selon l'envie de l'utilisateur . De même pour le nombre de bombes. Nous avons décidé de mettre 4 attributs:

- *width*: La largeur
- *height*: La hauteur
- *nbrBomb*: Le nombre de bombes que le plateau aura
- *grid*: Un conteneur à 2 dimensions qui représente le plateau lui-même.

Biensur que ce n'est pas la seule façon d'implémenter un plateau, nous aurions pu aussi choisir de mettre un tableau de booléen qui contiendrait les position des bombes.

Case

Cette classe représente une case dans le plateau. En fait, l'attribut *grid* de la classe **Board** est un vecteur de 2 dimensions, dont chaque élément est une **Case**. Une case est composée de plusieurs éléments:

-*row*: La ligne correspondante

-*column*: La colonne correspondante

-*state*: Une énumération de 3 états qu'une case peut prendre: cachée (*HIDDEN),* révélée (*REVEALED)* , ou marquée (*MARKED).*

-Toutes les cases sont cachées au début du jeu. Lors que l'utilisateur choisi une position (ligne et colonne) pour révéler la case, ou s'il clique dessus si on est en mode graphique, son état passe à révélé (REVEALED), et s'il pense qu'une case contient une bombe, il peut choisir de la marquer, et un drapeau apparaîtra dans cette case. Une case marquée ne peut pas être révélée ni par l'utilisateur, ni par les autres cases lors de la vérification des voisins.

-*containsBomb*: Booléen qui sera à true si une case contient une bombe ou false sinon.

-*nbBombNeighbours*: Le nombre de cases voisines qui contient une bombe.

Au départ nous mis les attributs *row* et *column* dans une classe **Position**, mais nous avons trouvé que ce n'était pas nécessaire, vu que cette classe ne contiendrait que 2 variables du type unsigned.

Game

C'est cette classe qui va créer les éléments nécessaires pour que le jeu se déroule. elle sera composée de:

- *board*: Un plateau
- *player*: Le joueur courant
- *time:* Le temps de la partie.

Nous allons aussi rajouter un timer qui sera affiché lors du déroulement de la partie.

Player

Cette classe représente le joueur courant. Elle contient 2 attributs:

-*username:* Le pseudo du joueur
-*score*: Le score qu'il accomplira lors de sa session de jeux.

Il y aura un *hall of fame* dans lequel se trouvera les meilleurs joueurs, c'est à dire ceux qui ont eu les meilleurs scores, en finissant le jeu en moins de temps et de tours possibles. La notion de tour n'est pas présente dans toutes les implémentations du démineur, mais nous avons choisi de la mettre pour la notre. Ce sera expliqué tout de suite.

Score

Cette classe représente le score du joueur. Un score est déterminé selon le nombre de tours, c'est à dire le nombre de fois que l'utilisateur révèle une case, et le temps de la partie. Elle comprend:

-*nbTours*: Le nombre de tours comme expliqué ci dessus
-*time*: Le temps de jeux en seconde

Nous ne sommes pas encore sûr si le temps sera en secondes, puis transformé en HMS pour l'affichage dans le *hall of fame*, ou si ça sera implémenté dans une classe Time à part avec des attributs et méthodes indépendantes.

GameView

GameView va permettre d'afficher le jeux en cours, et donc le plateau, le temps qui découle et le joueur courant.

Comme dit au début de ce rapport, il est possible qu'il y ait des petits changements par la suite. Donc il est possible que l'implémentation de cette classe, et de la classe **Game** change quand nous serons en train de faire la partie graphique, où il faudra implémenter le design pattern MVC (*Modèle/Vue/Contrôleur*).

Modifications (Remise console)

Nous avons dû faire des modifications par rapport à la première remise. Nous allons les présenter ci dessous.

Classe Game

- ~~*time*: Le temps de la partie.~~
- *isGameOver*: Si le jeu est fini ou pas.

-Nous avons décidé de mettre le temps dans la classe Player, qui sera le temps qu'il aura pris pour finir une partie. Nous avons utilisé std::chrono dans la méthode principale directement pour la remise console. Pour la remise graphique, ce sera un timer qui s'affichera en temps réel, donc c'est possible qu'on remettra un attribut temps dans la classe Game.

-Les 3 méthodes d'action dans le jeu (reveal, markCase et removeMarkCase) ont été déplacées de la classe Board vers cette classe-ci pour qu'on puisse jouer à partir d'ici, vu que cette classe est la façade.

-Une méthode endGame à été rajoutée . Elle permet de terminer le jeux si le joueur a gagné, c'est à dire s'il a révélé toutes les cases.

-L'attribut *isGameOver*, qui nous permet d'identifier si le joueur a révélé une case avec une bombe, à été ajoutée.

-Ajout d'une méthode *bombFirstReveal()* qui nous permet de gérer le fait que la première case révélée par le joueur ne soit jamais une bombe.

-Game possède 2 constructeurs désormais. Un avec seulement le nom du joueur (paramètres par défaut) et un autre avec la possibilité de choisir la taille du plateau,

Classe GameView

Cette classe contient désormais plusieurs méthodes d'affichage. Une méthode pour afficher le joueur, deux pour le plateau, une pour la case, et une pour le temps , et pour finir, la méthode update de observer.

Pour le plateau, une méthode l'affiche sans les bombes, et l'autre affiche avec les bombes lorsque le jeu est fini.

Nous avons implémenté le Design Pattern Observateur/Observé.

Classe Player

-Comme nous avons supprimé la classe *Score*, nous avons mis le temps et le nombre de tours accomplis dans player directement par question de facilité.

-Ajout de la méthode *incrementTour()* qui va incrémenter l'attribut *nbTour* a chaque fois que le joueur révèle une case.

Classe Board

-Ajout de la méthode *isInsideGrid()* pour vérifier si une position reçue se trouve bien dans le plateau, c'est-à-dire une position qui n'est pas en dehors du plateau.

-Ajout de méthode *random* pour générer des nombres aléatoires qu'on utilise lors de l'initialisation des bombes.

-Ajout d'une méthode *bombProba()* qui nous permet de placer les bombes en fonction du pourcentage qu'on reçoit en paramètre. La valeur reçue corresponde à la probabilité qu'une case aura d'avoir une bombe.

-Ajout d'un deuxième constructeur si le joueur ne souhaite pas choisir le nombre de bombes.

-Ajout de la méthode *initBombManual()* ET *countBombs()* qui seront utiles pour nos tests unitaires.

Classe Case

-Nous avons enlevé les attributs row et column, car on les reçoit désormais directement par paramètre (plus pratique).

Fichiers ajoutées

Input.hpp - pour gérer les saisies du joueur en mode console.

Observer.hpp et Subject.hpp - Nécessaire pour le design pattern Observateur/Observé.

Autres modifications

-Notre projet possède désormais le template “Subdirs”, ce qui rend chaque plus propre.

-Des tests unitaires ont été rajoutés (framework Catch).

-Nous avons déplacé toutes les méthodes d’affichage vers la partie “console”. Donc il n’y a plus d’affichage dans le modèle.

Dernière remise (graphique)

Comme promis, nous avons fait le hall of fame pour cette remise. Les ajouts étaient:

- **Score**: Contient le score du joueur, avec son nom, la hauteur et largeur du tableau choisies, le nombre de bombes choisies, et le temps qu'il a pris pour finir le jeu.
- **Hall Of Fame**: Lecture/écriture des scores des joueurs dans un fichier texte (classement des 5 meilleurs joueurs)

Nous avons déplacé le **Time**, qui était dans la console avant, dans la classe Game, et on l'initialise dans le constructeur de celle-ci.

Pour la partie graphique, nous avons utilisé l'interface de QT Creator pour les éléments graphiques. Les classes ajoutées sont:

- **ConfigGame**: Représente une fenêtre qui contient les éléments nécessaires pour créer une partie comme le choix du nom du joueur, la hauteur et la largeur du plateau, ainsi que le nombre des bombes. Pour ce dernier, deux onglets sont disponibles, un pour un montant fixe de bombes, et l'autre pour un montant en pourcentage, qui sera calculé par rapport à la taille du plateau. Si le joueur ne remplit aucun de ces champs, une partie avec des valeurs par défaut est lancé. Les valeurs par défaut sont
    - plateau 9x9
    - 10 bombes
    - pseudo "Guest"

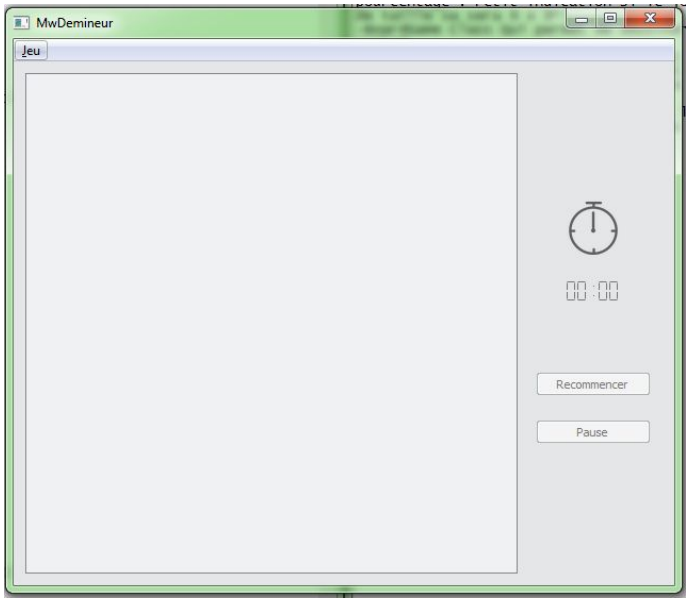
- **BoardGame**: Permet de dessiner le plateau du jeux à l'interface graphique, et gère les événements de la souris qui sont soit un clic gauche pour révéler une case, soit un clic droit pour marquer une case.

- **HallOfFameGui**: Permet de lire et d'afficher le hall of fame avec les 5 meilleurs joueurs pour une taille de plateau et un nombre de bombe spécifiques. Le hall of fame est accessible soit automatiquement quand un joueur gagne une partie, soit en cliquant sur le bouton "Meilleur temps" au côté droit de la fenêtre de l'application, juste en bas du timer.

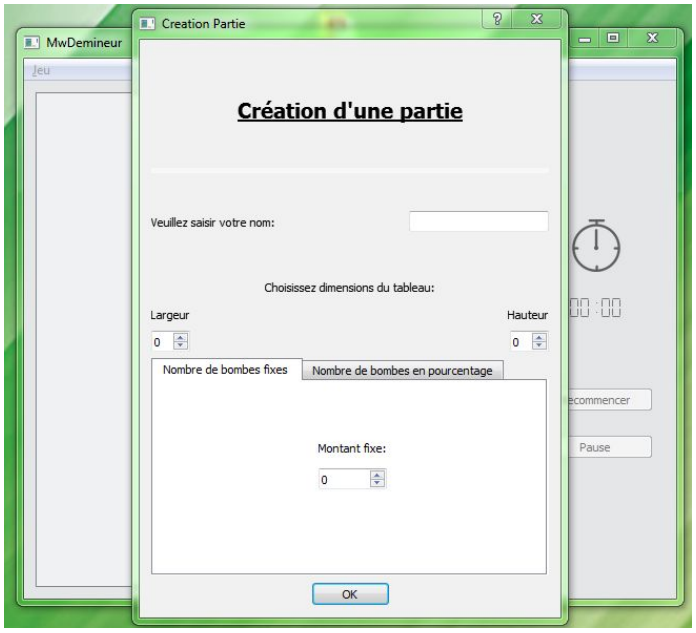
- **MwDemineur**: C'est la fenêtre principale du jeu, qui s'ouvre lorsqu'on démarre l'application. Elle contient 1 menu "Jeu", que lui contient 2 menus:
    - Nouveau: Créer une nouvelle partie (appelle la classe ConfigGame pour choisir les données pour la partie)
    - Quitter: Quitte l'application. Cela a le même effet que cliquer sur la croix pour fermer la fenêtre.
    - Le menu "Fermer" remplace le menu quitter si une partie est en cours.
- Un des attributs de MwDemineur est un BoardGame, qui affiche le plateau du jeu ainsi que le timer, et les boutons pour les différentes interactions. Les boutons sont:
    - **Recommencer**
        - Recommence la partie avec les mêmes configurations, donc pas besoin de tout retaper si on veut recommencer avec exactement les mêmes paramètres.
    - **Pause**
        - Pause la partie en cours. Le plateau se cache, et réapparaît quand on reprend le jeu.
    - **Start**
        - Reprend la partie de la pause.
    - **Meilleur temps**
        - Ouvre une nouvelle fenêtre avec le hall of fame.

Screenshots

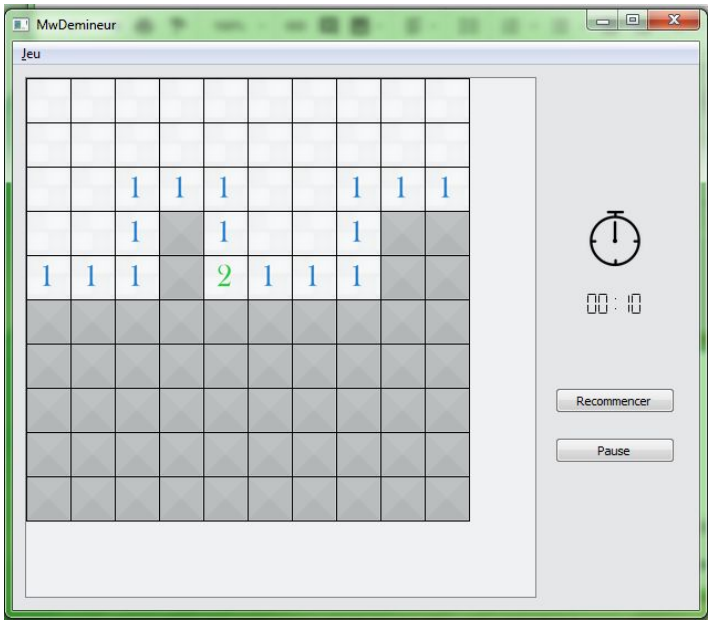
La première fenêtre à s'ouvrir:



Quand on va sur "Jeu" -> "Nouveau":



Une partie en cours:



Quand le jeu est en pause:

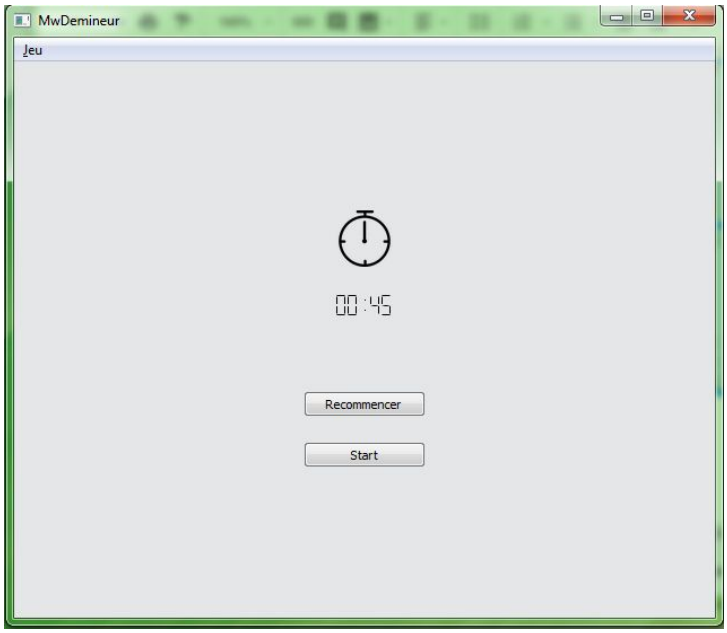
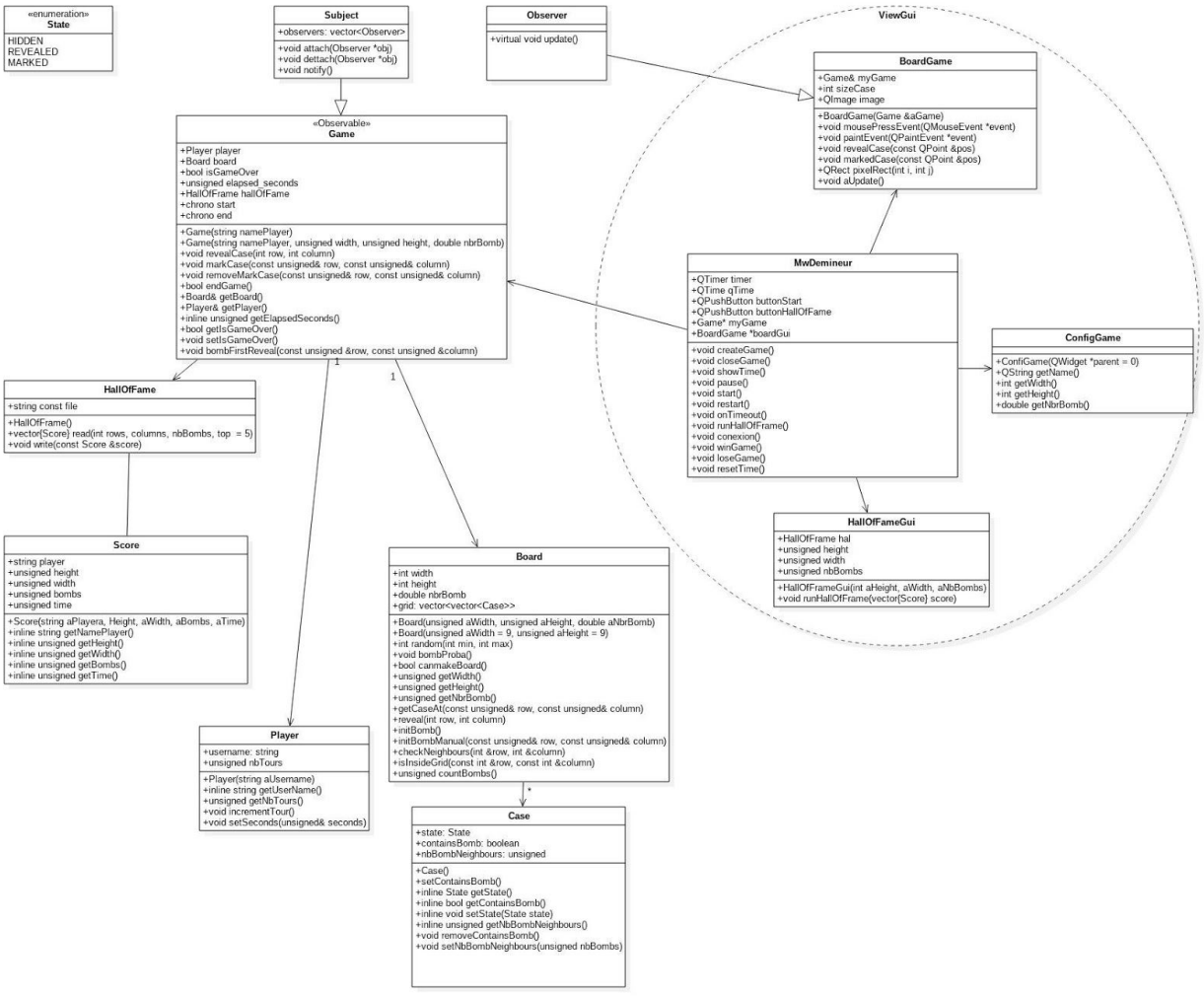


Diagramme de classes (remise graphique)



Conclusion

Nous arrivons enfin à la fin de ce projet. Nous pouvons constater sans aucun doute que notre expérience avec le langage c++ à beaucoup augmenté. Notre projet est loin d'être parfait, mais nous sommes très content de notre travail. En ayant 3 projets à faire au cours de l'année, nous pouvons confirmer que ce n'était pas facile de gérer le temps, mais je suis certain que nous avons fait de notre mieux, et nous avons appris beaucoup de choses au cours de ce projet, non seulement l'expérience avec le langage, mais aussi à travailler en équipe, ce qui peut paraître facile, mais ce n'est pas toujours évident. On tient à remercier également notre maître-assistant, M. Romain Absil, qui nous a suivi parmis ce parcours, et nous a guidé vers le meilleur chemin.

Bibliographie

- Eléments de rédaction scientifique en informatique, de Hadrien Mélot (Service d'Algorithmique, Institut d'Informatique Faculté des Sciences, UMONS) hadrien.melot@umons.ac.be
  Mars 2011, Version 3

- Wikipédia (https://en.wikipedia.org/wiki/Minesweeper_(video_game))
- Les documents concernant l'énoncé du projet sur poÉSI