INFO0030 : Projet de Programmation
Démineur

B. Donnet, E. Marechal
Université de Liège

1 Contexte

Le *démineur* est un jeu de réflexion dont le but est de localiser des mines cachées dans un champ virtuel avec pour seule indication le nombre de mines dans les zones adjacentes. [1]

Le jeu se présente sous la forme d'un tableau de cases à deux dimensions. Certaines cases contiennent des mines. Au départ toutes les cases sont masquées (et on ne voit donc pas les mines). L'objectif est de localiser les mines (le nombre de mines restant à localiser étant donné au joueur). Le joueur peut au choix découvrir le contenu d'une case, ou la marquer pour signaler la présence d'une mine lorsqu'il pense en avoir localisé une. Les cases découvertes affichent le nombre de mines dans les 8 cases voisines, sauf si la case contient elle même une mine (auquel cas le joueur meurt et la partie est terminée). Si le joueur n'est pas mort, la partie est gagnée lorsque toutes les cases sont découvertes ou marquées par un drapeau, et que le nombre de drapeaux est égal au nombre de mines. Une capture d'écran du démineur, sous Windows, est donné à la Fig. 1.

Le joueur doit terminer le jeu endéans un certain temps. Si, une fois le temps écoulé, le joueur n'a pu trouver toutes les mines, le jeu est terminé et le joueur a perdu. Dans le cas contraire, le joueur a gagné et son score est le temps restant au moment où il gagne.

FIGURE 1 – Le démineur, version windows (source : `http://www.toucharger.com`)

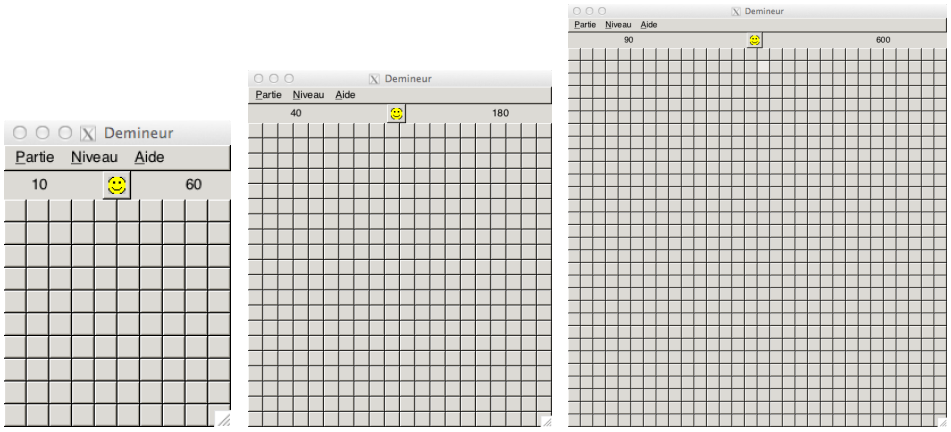
Dans ce projet, vous devrez, en **binôme**, implémenter une version du Démineur en suivant le pattern **Modèle-Vue-Contrôleur** (MVC) étudié durant le cours théorique.

2 Dimensionnement du Jeu

Par défaut, votre plateau de jeu devra mettre en place un carré de 10 × 10 carrés comptant 10 mines. Pour ce plateau "par défaut", le joueur disposera de 60 secondes pour terminer le jeu.

Il est permis, à l'utilisateur, de modifier la composition du plateau de jeu. Pour ce faire, deux possibilités :

1. Source : wikipedia.org – cfr. `http://fr.wikipedia.org/wiki/D\'{e}mineur_(jeu)`



(a) Débutant (b) Intermédiaire (c) Expert

FIGURE 2 – Les trois niveaux standards de difficulté du jeu

— en ligne de commande. Dès le lancement du programme, l'utilisateur spécifie le format du jeu (cfr. Sec. 9 pour les détails).
— via le menu du jeu. Dans ce cas, trois choix s'offrent à l'utilisateur :
  — *Débutant*. Il s'agit du plateau de jeu "par défaut". Il est illustré à la Fig. 2(a).
  — *Intermédiaire*. Le plateau se présente alors comme un carré de 20×20 carrés comptant 40 mines. Le joueur dispose de 3 minutes pour terminer le jeu. Il est illustré à la Fig. 2(b)
  — *Expert*. Le plateau se présente alors comme un carré de 30×30 carrés comptant 90 mines. Le joueur dispose de 5 minutes pour terminer le jeu. Il est illustré à la Fig. 2(c).

Il est bon de noter que si l'utilisateur choisit de spécifier la dimension du jeu en ligne de commande, il ne peut pas faire ce qu'il veut. Au maximum, la hauteur du plateau sera de 30 carrés et la largeur de 30 carrés. Au maxium, un plateau peut compter 900 mines. Le temps maximum pour terminer le jeu est de 5 minutes. Toute configuration dépassant un de ces éléments doit se voir proscrite ou bien ramenée à la taille standard. [2]

Par défaut, les cases du plateau de jeu ont une taille de 20×20 pixels.

3 Etapes Importantes du Jeu

L'exécution du jeu nécessite plusieurs étapes :

1. *Initialisation*. Durant cette étape, les éventuels paramètres en ligne de commande sont parsés et, dans tous les cas, un tirage aléatoire du tableau à deux dimensions a lieu. Ce dernier élément permet de déterminer des emplacements aléatoires pour les mines, dans le plateau de jeu (cfr. Sec. 4).
2. *Interface Graphique*. L'interface graphique (IHM) du jeu est chargée (cfr. Sec. 5).
3. *Découvrement Case*. Le joueur a cliqué (à l'aide du bouton gauche de la souris) sur une case. Il s'agit de déterminer si la case cache ou non une mine. Si la case cache une mine, la mine explose et le jeu se termine (le joueur a perdu). Dans le cas contraire, il faut décompter le nombre de mine alentours (cfr. Sec. 4) et afficher ce nombre sur la case découverte par le joueur. Si ce nombre est zéro, il faudra répéter l'opération pour les huit cases alentour (opération à réitérer tant que l'on trouve des cases sans mines alentour). On affichera le nombre total mines restantes. [3]
4. *Pose Drapeau*. Le joueur sélectionne, avec le bouton droit de la souris, une case (non encore découverte) et y pose un drapeau (supposant, de fait, la présence d'une mine sous la case).
5. *Découvrement rapide*. Lorsque le nombre de drapeaux entourant une case découverte correspond au nombre inscrit sur cette case, le joueur peut en un clic (à l'aide du bouton central) découvrir toutes les cases alentour qui ne portent pas de drapeau

2. Dans ce dernier cas, un message doit informer l'utilisateur de son erreur.
3. Des images représentants les chiffres de 1 à 8 sont disponibles sur la page Web du cours

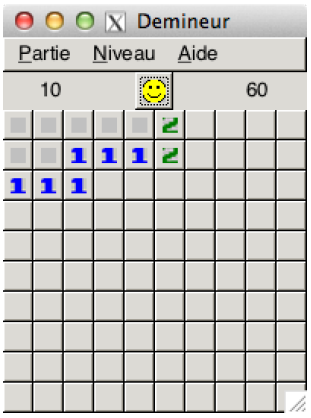


FIGURE 3 – Exemple quand on clique sur le coin supérieur gauche et qu'il n'y a pas de mines dans le voisinage de cette case

4 Gestion des Mines

4.1 Placement des Mines

Pour pouvoir placer des mines aléatoirement dans le plateau de jeu, il faut utiliser un générateur de nombres aléatoires (cfr. Sec. 6.2). Si le plateau est de dimension $n \times m$, il suffit de tirer aléatoirement un chiffre entre 0 et $n-1$ afin d'obtenir le numéro d'une ligne du plateau. Ensuite, on réitère le tirage aléatoire mais, cette fois, entre 0 et $m-1$, ce qui nous donnera le numéro d'une case sur la ligne choisie.

Il suffit de répéter ce schéma jusqu'à ce qu'on ait placé toutes les mines.

Cet algorithme est assez simple à implémenter mais fortement inefficace dans les cas où le plateau de jeu est plein, voire très plein. Si par exemple on décide de jouer sur une grille de dimension 20×20 et qu'on veut y mettre 361 bombes. Quand on aura déjà placé 360 bombes et qu'on débutera l'algorithme pour placer la dernière, il y a de fortes chances (90%) que la case tirée au hasard soit déjà occupée par une bombe et qu'on doive refaire le tirage au sort de nombreuses fois.

Il est donc souhaitable, dans le cadre de ce projet, de réfléchir à un algorithme plus efficace.

4.2 Comptage des Mines dans le Voisinage

Une fois le plateau de jeu rempli avec les mines, il faut pouvoir garder une trace du nombre de mines entourant chaque case.

Une façon simple de faire est de reparcourir entièrement le plateau de jeu, case par case. Pour chacune d'entre elles, on compte le nombre de mines dans le voisinage direct, et lui assigne ce nombre. Une structure de données supplémentaire est donc nécessaire pour retenir ce nombre.

Une version améliorée de cette algorithme effectue le comptage lors du placement des mines.

4.3 Découvrement du Voisinage

Lorsque l'on clique sur une case vide (qui ne contient pas de mine), il faut compter le nombre de mines voisines de la case pour pouvoir l'afficher. Pour un plateau de jeu de dimension $n \times m$, une case de coordonnées (x, y) (avec $0 \leq x \leq n-1$ et $0 \leq y \leq m-1$) dispose de huit cases voisines.

La partie complexe du démineur est la propagation des cases sûres : si une case sans mine n'a aucun voisin avec une mine, on peut automatiquement découvrir les cases voisines, etc. C'est ceci qui permet d'obtenir la configuration illustrée à la Fig. 3 en un seul clic sur le coin supérieur gauche.

On parle de *composante connexe* : on découvre toutes les cases qui ne contiennent rien (ni mine, ni un entier strictement supérieur à 0), en s'arrêtant aux cases qui contiennent un entier strictement supérieur à 0.

Une approche récursive permet de découvrir, élégamment, le voisinage d'une case vide.

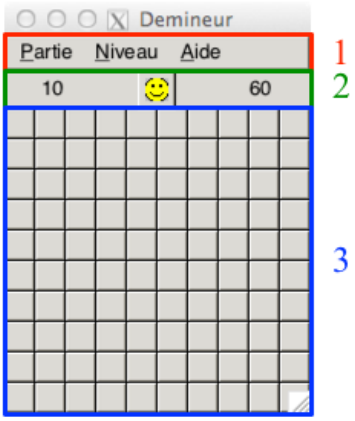


FIGURE 4 – Interface graphique générale du jeu

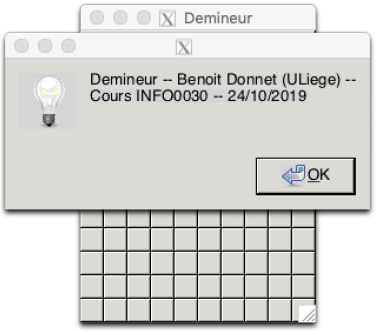


FIGURE 5 – Boîte de dialogue "A Propos"

Le calcul de la composante connexe au démineur se fait de la manière suivante pour une case donnée (sans mine) :

— si la case était déjà découverte, on s'arrête
— sinon :

1. on compte le nombre de mines voisines, `c`
2. on met à jour cette case dans le plateau avec la valeur `c`
3. on teste
   — si la case a au moins une mine parmi ses voisine, on s'arrête
   — si la case n'a aucune mine parmi ses voisines, on recommence la fonction récursivement sur toutes les cases voisines.

Cette fonction s'arrête forcement au bout d'un moment car on ne fait pas d'appel récursif si la case de départ était déjà découverte. Comme toutes les cases qu'on explore sont découvertes, il ne peut pas y avoir un nombre infini d'appels récursifs.

5 Interface Graphique

La Fig. 4 présente l'IHM générale du jeu. Comme indiqué sur la figure, l'IHM est divisée en trois parties :

1. *Menu*. Le menu comprend trois sous-menu :
   (a) *Partie*. Ce menu permet de démarrer une nouvelle partie ou bien quitter le jeu.
   (b) *Niveau*. Ce menu permet de sélectionner un des trois niveaux du jeu (cfr. Sec. 2).
   (c) *Aide*. Ce menu permet d'obtenir via un item unique, *A propos*, ouvrant une boîte de dialogue des informations relatives au créateur. Un exemple de boîte de dialogue est donné à la Fig. 5.
2. *Plateau d'information*. Ce plateau indique le nombre de mines restantes (à gauche), le temps écoulé (à droite) et un bouton indiquant, à tout moment, la situation du joueur (jeu en cours de route, jeu perdu, jeu gagné). Une image particulière permet d'illustrer chacune de ses situations. [4]
3. *Plateau de Jeu*. Ce plateau contient les cases qui cachent (ou pas) les mines.

6 Eléments Additionnels C

Cette section vous présente différents éléments additionels du C et de GTK qui peuvent vous être utiles pour l'implémentation du projet.

4. Les différentes images nécessaires au jeu sont disponible sur la page Web du cours.

6.1 GTK

6.1.1 Dimensionnement des Boutons

Une fois un bouton créé (et, plus généralement, n'importe quel élément de type `GtkWidget *`), on peut toujours modifier sa taille par défaut. Il suffit d'utiliser la procédure suivante :

```
void gtk_widget_set_size_request(GtkWidget *wgt, gint largeur, gint hauteur);
```

qui redimensionne le widget `wgt` en un widget de `largeur` × `hauteur`. L'unité de mesure est ici le pixel.

6.1.2 Image dans un Bouton

Une façon simple d'ajouter une petite image dans un bouton est de considérer le bouton comme un container. Cela se fait comme suit :

```
char *chemin = "Mon_Bouton.png";

GtkWidget *pBouton = gtk_button_new();

GtkWidget *pImage = gtk_image_new_from_file(chemin);
gtk_container_add(GTK_CONTAINER(pBouton), pImage);
```

6.1.3 Gestion des Boutons de la Souris

Les étapes 3, 4 et 5 décrites à la Sec. 3 correspondent à trois actions différentes de l'utilisateur avec la souris : clic droit, clic milieu et clic gauche.

Il faut donc pouvoir récupérer le numéro du bouton de la souris cliqué, à l'intérieur de la fonction de callback associée aux cases du tableau. Pour cela, il ne faut pas utiliser le signal "clicked", mais le signal "button_press_event". La fonction de callback reçoit alors trois paramètres :

— le widget ayant émis le signal ;
— l'événement survenu sur ce widget (du type `GdkEventButton *`) ;
— la donnée utilisateur (`userdata`).

Le type `GdkEventButton` est un type structuré contenant le champ `button`, qui vaut 1, 2 ou 3 selon le bouton de la souris qui a provoqué l'envoi du signal (1 pour le bouton gauche, 2 pour le bouton du milieu et 3 pour le bouton droit).

Pour que ce numéro de bouton soit effectivement enregistré et transmis à la fonction de callback, il faut le demander à GTK, en appelant, avant de lancer l'interface, la fonction `gtk_widget_set_events`. Par exemple, pour que le callback d'un bouton `b1` puisse recevoir le numéro de bouton de la souris, on appellera :

```
gtk_widget_set_events( b1, GDK_BUTTON_PRESS_MASK );
```

6.1.4 Gestion des Timers

Dans le Démineur, le joueur dispose d'un certain laps de temps pour terminer le jeu. Si à la fin du temps imparti, il n'a pas découvert toutes les bombes, le jeu se termine et il a perdu.

Il existe une façon très simple en GTK pour appeler une fonction spécifiée au bout d'un certain délai. Par exemple la fonction `tic` ci-dessous sera appelée toutes les 1s, et recevra à chaque fois le pointeur vers une structure de données que vous utilisez (ici, `My_Data_Structure`).

```
gint tic(gpointer data){
   My_Data_Structure *mds = (My_Data_Structure*)data;
   printf( "tic\n" );

   //Reenclenche le timer.
   g_timeout_add(100, tic, (gpointer)mds);

   return 0;
 }//end tic()

```

```
//...
//Enclenche le timer pour se declencher dans 20ms.
g_timeout_add(20, tic, (gpointer)mds);
```

Pour simplifier, on va supposer que le label pour le timer a été placé dans la variable suivante : `GtkWidget *value_timer`.

En utilisant `gtk_label_set_text(GTK_LABEL(value_timer), str)`, vous remplacez le texte stocké dans ce label par le contenu de la chaine de caractère `str`. Votre fonction tic doit décrémenter votre compteur et l'afficher. Lorsqu'il arrive à zéro, le jeu se termine et le joueur a perdu. Le bouton central de la zone 2 change alors en chargeant l'image indiquant que le joueur a perdu.

6.2 Contrôler l'Aléatoire

En C, on peut produire une suite pseudo-aléatoire d'entiers en utilisant la fonction `rand()` de la bibliothèque standard (`stdlib.h`). A chaque fois qu'un programme appelle cette fonction, elle retourne l'entier suivant dans la suite. Les entiers de la suite sont compris entre `0` et `RAND_MAX` (une constante prédéfinie).

Par exemple, le programme suivant :

```
#include <stdlib.h>
#include <stdio.h>

int main(){
  int i;

  for(i=0; i<10; i++){
    int a = rand();
    printf("%d\t", a);
  }//fin for - i

  printf("\n");

  return EXIT_SUCCESS;
}//fin programme
```

produit la sortie suivante :

```
16807    282475249        1622650073       984943658        1144108930
470211272        101027544        1457850878       1458777923       2007237709
```

Si on cherche à tirer àléatoirement un nombre entre 0 et N (compris), il suffit d'utiliser le reste de la division par N :

```
int x = rand()%N;
```

Si on veut tirer un entier entre A et B compris, on peut faire :

```
int x = rand() % (B-A+1) + A;
```

Pour produire un flottant entre A et B, on peut faire :

```
double x = (double) rand() / RAND_MAX * (B-A) + A ;
```

La suite produite par `rand()` n'est pas vraiment aléatoire. En particulier, si on lance le programme précédent plusieurs fois, on obtiendra exactement la même suite.

Pour rendre le résultat apparemment plus aléatoire, on peut initialiser la suite en appelant la fonction `srand()` (définie aussi dans `stdlib.h`), à laquelle on passe en paramètre une valeur d'initialisation. La suite produite par `rand()` après cet appel dépendra de la valeur que l'on passe à `srand()` lors de l'initialisation. Une même valeur d'initialisation produira la même suite.

Par exemple, le programme suivant :

```
#include <stdlib.h>
#include <stdio.h>

int main(){
```

```
  int i;
  srand(2);

  for(i=0; i<5; i++){
    int a = rand();
    printf("%d\t", a);
  }//fin for - i

  printf("\n");
  srand(2);

  for(i=0; i<5; i++){
    int a = rand();
    printf("%d\t", a);
  }//end for - i

  printf("\n");

  return EXIT_SUCCESS;
}//fin programme
```

produit la sortie suivante :

```
33614   564950498       1097816499      1969887316      140734213
33614   564950498       1097816499      1969887316      140734213
```

Pour que la suite soit vraiment imprévisible, on peut passer à `srand()` une valeur dépendant de l'heure. Par exemple en utilisant la fonction `time()` de la librairie standard :

```
srand(time(NULL));
```

7 SCM

Dans le cadre de ce projet, il vous est aussi demandé d'utiliser un SCM (source code manager). L'utilisation du SCM au sein de votre binôme interviendra dans la note globale de votre projet. Vous devrez, en outre, décrire la façon dont vous avez utilisé le SCM dans votre rapport (cfr. Sec. 8).

Le type de SCM à utliser vous est imposé. Il s'agit d'un Git-Hub. Il est disponible à l'adresse suivante : `https://gitlab.montefiore.ulg.ac.be/users/sign_in`.

Les détails relatifs aux SCMs disponibles (et comment se créer un compte) sont disponibles sur la page web du cours.

L'utilisation du Git-Hub nécessite une inscription préalable. Les deux membres du binôme doivent s'inscrire. Une fois que c'est fait, un des deux membres du binôme a la possibilité de créer un projet. Le nom du projet, dans le Git-Hub, doit **obligatoirement** suivre la nomenclature suivante : `INFO0030_GroupeXX`, où `XX` désigne votre numéro de groupe (il s'agit donc bien d'un nombre à deux chiffres). Le créateur du projet est alors libre d'ajouter l'autre membre du binôme au projet nouvellement créé. Une fois cette étape passée, les deux membre du binôme (et uniquement eux) peuvent accéder au Git-Hub, au Wiki associé, etc.

L'URL associée à votre projet est disponible dans la partie supérieure droite de l'écran (une fois, bien entendu, que le projet a été créé). L'étape suivante consiste donc à cloner le projet en local sur votre machine en utilisant la commande

```
$>git clone <URL>
```

où `<URL>` désigne l'URL de votre projet.

Pour rappel, voici quelques commandes qui pourraient vous être utiles :

```
$>git add <fichier>
```

ajoute le fichier `<fichier>` à votre projet.

```
$>git commit -m "Log Message"
```

permet de répercuter les changements locaux sur le serveur du Git-Hub.

```
$>git pull
```

permet de mettre à jour la copie locale.

8 Rapport

En plus du code source de votre application, nous vous demandons de joindre un rapport détaillé et clair expliquant :

- l'architecture générale de votre code. Quels sont les grands concepts de votre code et comment ils interagissent entre eux.
- les structures de données. Vous serez amenés, dans ce projet, à développer des structures de données. Décrivez les dans le rapport et pensez à discuter la pertinence et/ou le coût de ces structures.
- les algorithmes particuliers. Vous pourriez être amenés, dans ce projet, à développer des algorithmes poussés, notamment pour le déroulement du jeu. Décrivez l'idée de ces algorithmes dans votre rapport et comment vous les avez implémentés (structures de données particulières, fonctionnement général, ...). [5]
- un profiling et une analyse de performance de votre code. En particulier, vous veillerez à bien étudier les performances des algorithmes importants en mesurant les temps d'exécution en fonction de diverses situations de jeu. Vous procéderez de même pour les structures de données, en particulier en essayant d'estimer la quantité mémoire utilisée. [6]
- l'interface graphique du jeu. Fournissez des captures d'écran et commentez les. Expliquez comment vous avez organisé votre jeu (table, box, ...).
- la gestion du code. Expliquez comment vous avez géré le SCM et ce que cela vous a apporté tout au long du projet.
- la coopération du sein du groupe. Expliquez comment votre binôme a fonctionné et comment vous avez géré votre coopération.
- les améliorations possibles. Décrivez les améliorations possibles à votre application (par exemple, si vous aviez disposé d'un mois supplémentaire).
- les éléments que vous avez appris. Décrivez ce que ce projet vous a apporté et ce que vous en avez appris.

Votre rapport doit être rédigé en LaTeX. [7] Le document LaTeXdoit suivre le template donné sur la page web du cours. Vous penserez à prendre en compte les éléments abordés lors de la formation relative à la communication écrite.

Vous veillerez à joindre, dans votre archive (cfr. Sec. 9), les sources de votre rapport ainsi qu'une version PDF.

9 Enoncé du Projet

Contrairement aux autres projets, celui-ci doit être réalisé par binôme. A chaque groupe a été associé un identifiant (unique) sous la forme d'un nombre naturel à deux chiffres.

Il vous est demandé d'écrire un programme implémentant le jeu du Démineur tel que décrit dans ce document.

Votre projet devra :

- être soumis dans une archive `tar.gz`, appelée `demineur_XX.tar.gz`, via la plateforme de soumission, où `XX` désigne votre identifiant de groupe. [8] La décompression de votre archive devra fournir tous les fichiers nécessaires dans le répertoire courant où se situe l'archive. Vous penserez à joindre tous les codes sources nécessaires ainsi que votre rapport.

5. Il n'est pas nécessaire d'indiquer les invariants, les spécifications formelles et, plus généralement, l'approche constructive dans votre rapport.
6. Vous veillerez à appliquer les principes que vous avez vus au cours MATH1472 (Statistique Descriptive).
7. Nous vous renvoyons à votre formation LaTeXreçue en début de quadrimestre.
8. Pour rappel, l'adresse de la plateforme de soumission est `https://submit.montefiore.ulg.ac.be`. N'oubliez pas de vous enregistrer en tant que groupe !

— définir les fonctionnalités nécessaires à la mise en place de l'IHM du Démineur. Des fichiers `png` vous sont fournis en supplément de façon à implémenter le jeu.
— être modulaire, i.e., nous nous attendons à trouver un (ou plusieurs) header(s) et un (ou plusieurs) module(s). De manière générale, votre code devra impérativement suivre le pattern **modèle-vue-contrôleur** (MVC) tel que vu au cours. [9]
— être parfaitement documenté. Vous veillerez à spécifier correctement chaque fonction/procédure/structure de données que vous définirez. Votre documentation suivra les principes de l'outil doxygen.
— appliquer les principes de la programmation défensive (vérification des préconditions, vérification des mallocs, ...). Pensez à libérer la mémoire allouée en cours de programmation afin d'éviter les fuites de mémoire.
— utiliser un SCM (cfr. Sec. 7).
— contenir un rapport rédigé en LaTeX. Vous veillerez à fournir les sources de votre rapport ainsi que la version compilée en PDF (voir Sec. 8 pour le format attendu).
— comprendre un Makefile permettant au moins de
  — compiler votre programme. La commande

```
$>make demineur
```

  doit pouvoir compiler votre programme et générer un binaire exécutale appelé `demineur`.
  — générer la documentation, au format HTML, en suivant les principes de doxygen. La commande

```
$>make doc
```

  doit permettre la générati o n de la documentation dans un sous-répertoire `doc/`.
  — générer le PDF de votre rapport. La commande

```
$>make rapport
```

  doit permettre cela.
— pouvoir exécuter le binaire `demineur` de la façon suivante :

```
$>./demineur -l -h -t -m [-H]
```

  où
  — `-l` : largeur du plateau (entre 10 et 30).
  — `-h` : hauteur du plateau (entre 10 et 30).
  — `-t` : temps pour résoudre le problème (entre 60 et 600).
  — `-m` : nombre maximum de mines (entre 10 et 900).
  — `-H` : aide pour la ligne de commande (optionnel).

Si nous appliquons la commande suivante à votre archive :

```
$>tar xvfz demineur_XX.tar.gz
```

l'architecture suivante de répertoires doit apparaître :

```
GroupeXX
  |
  ---source/
  ---doc/
  ---rapport/
```

où le sous-répertoire `source/` contient les fichiers sources de votre programme ainsi que le Makefile, le sous-répertoire `doc/` contient la documentation au format HTML déjà générée par doxygen (pensez à bien mettre une règle Makefile pour que nous puissions la regénérer dans ce sous-répertoire) et, enfin, le sous-répertoire `rapport/` contient les sources de votre rapport et le fichier PDF correspondant.

Attention, le projet ne se termine pas à la soumission de votre archive. En plus de tout cela, vous devrez réaliser une démonstration de votre projet devant l'équipe pédagogique. Cela consistera en une exécution live (maximum 5 minutes par groupe) de votre application. Durant cette exécution, vous expliquerez,

9. cfr. Partie 3, Chapitre 3, du cours théorique

oralement, le contenu de votre rapport et vous démontrerez que votre jeu fonctionne parfaitement. Après les 5 minutes de démonstration, l'équipe pédagogique disposera (éventuellement) de 5 minutes afin de vous poser des questions pour clarifier certains points. L'ordre de passage pour la démonstration sera donné sur la page web du cours.

10 Cotation

Etant donné le caractère inhabituel et plus complet de ce projet par rapport aux autres, la grille de cotation que nous allons appliquer sera légèrement différente :

1. 50% de la note du projet sera affectée suivant la nomenclature classique adoptée dans le cours.
2. 10% de la note du projet portera sur la façon dont vous avez utilisé votre SCM.
3. 10% de la note du projet portera sur l'application du pattern MVC.
4. 15% de la note du projet portera sur le rapport écrit (qualité globale du document [10], orthographe, qualité et précision des explications, esprit de synthèse, ...).
5. 15% de la note du projet portera sur la démonstration live de votre jeu (qualité du jeu, clareté de la présentation, gestion du temps, scénarisation de la démonstration, réponses aux questions, ...).

10. Nous vous réferons à la formation que vous avez reçue en début de quadrimestre sur la façon de rédiger un rapport.