INFO0030 : Projet de Programmation
MASTERMIND

B. Donnet, K. Edeline, M. Goffart

Alerte : Évitez de perdre bêtement des points...

- Nous vous conseillons **vivement** de relire et d'appliquer les **guides de style et de langage**, mis à votre disposition sur eCampus (INFO0030, Sec. Supports pour le Cours Théorique). Cela vous permettra d'éviter de nombreuses erreurs et de perdre des points inutilement.
- Nous vous conseillons de consulter la **grille de cotation** utilisée pour la correction des projets afin d'éviter de perdre bêtement des points. Elle est disponible sur eCampus (INFO0030, Sec. Procédures d'Évaluation).
- Prenez le temps de lire attentivement l'énoncé. La plupart des réponses aux questions que vous vous posez s'y trouvent.
- Votre code sera compilé et testé sur les **machines CANDI qui servent de référence**. La procédure à suivre pour se connecter en ssh aux machines CANDI est disponible sur eCampus. Veillez donc à ce que votre code fonctionne dans cet environnement.
- Votre solution ne pourra pas être générée entièrement ou partiellement à l'aide d'un outil d'Intelligence Artificielle (e.g., ChatGPT, Blackbox, etc.).

1 Contexte

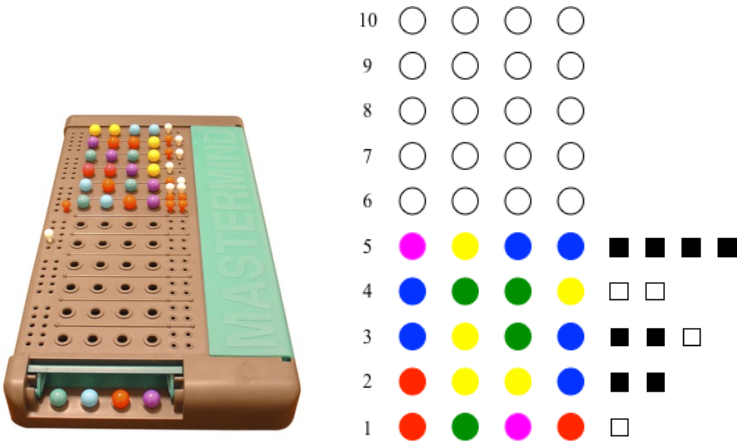
Le MASTERMIND est un jeu de société, de réflexion, et de déduction, inventé par Mordecai Meirowitz dans les années 1970. Il se joue à deux, avec un premier joueur qui place n pions de couleurs en une rangée ordonnée, sans que le deuxième joueur ne puisse les voir. Le but du jeu est alors pour ce dernier, de deviner les couleurs des pions et leur ordre en faisant un nombre limité j de propositions. Pour ce faire, il est aidé, à chacune de ses tentatives, par des indications données par le premier joueur sur ses choix de couleurs et de positions. [1]

Une partie se déroule typiquement de la façon suivante :

- le premier joueur choisi sa combinaison de n pions, et la dispose à l'abris du regard du deuxième joueur. Il peut choisir plusieurs pions de la même couleur dans une même combinaison. Ce joueur est appelé le *proposant*.
- le deuxième joueur fait sa première proposition, en choisissant à son tour une combinaison de n pions et en la présentant au regard du premier joueur. Ce joueur est appelé le *devinant*.
- le proposant indique au devinant le nombre de pions de sa proposition qui sont bien placés, et le nombre de pions de la bonne couleur mais qui sont mal placés. Cette indication de la part du proposant est appelée le *score*.
- le devinant peut alors faire une deuxième proposition, en s'aidant de ces indications.
- le proposant redonne le score correspondant à la nouvelle proposition et ainsi de suite jusqu'à ce que le devinant trouve la bonne combinaison ou qu'il ait épuisé ses j tentatives.

S'il trouve la bonne combinaison, le deuxième joueur gagne. Sinon, c'est le premier joueur qui remporte la partie. Un plateau de MASTERMIND est illustré à la Fig. 1. On voit que le plateau est divisé en trois parties :

1. Source : http://fr.wikipedia.org/wiki/mastermind



(a) Le vrai jeu (b) Un exemple de partie

FIGURE 1 – LE MASTERMIND

— une partie permettant au proposant de cacher sa combinaison ; Il s'agit d'un sous-ensemble de trous où le joueur va insérer les n pions de couleur (4 pions, dans la Fig. 1(a)) constituant la combinaison. La combinaison est invisible pour l'autre joueur ;
— une partie où le devinant va effectuer ses propositions successives ; chaque ligne est consituée d'un ensemble de trous dans lesquels le joueur insère les pions constituant sa proposition ; la proposition du devinant est donc de même nature que la combinaison chachée par le proposant ;
— une partie où le proposant indique au devinant le score de chaque combinaison proposée ; le score se présente sous la forme de pions rouges ou noirs (selon les versions de jeu) pour les couleurs bien placées, et de pions blancs pour les couleurs mal placées ; cela revient donc, comme indiqué supra, à indiquer le nombre de couleurs bien placées et le nombre de couleurs mal placées.

Il existe différentes variantes de MASTERMIND mais dans le cadre de ce projet, nous considérons qu'il y a sept couleurs différentes (vert, jaune, rouge, bleu, cyan, mauve, orange), que la combinaison à découvrir est composée de quatre pions et que le nombre maximum d'essais du devinant est dix. Le devinant a donc $7^4 = 2401$ choix possibles. Un exemple de partie où le devinant a réussi à trouver la solution en cinq coups est représenté à la Fig. 1(b).

2 Structure de Données

Vous allez dans un premier temps mener une réflexion autour des structures de données que vous utiliserez pour manipuler les différents éléments du jeu. Ces éléments devront être intelligemment représentés en mémoire tout au long du déroulement d'une partie.

Notamment, il vous faudra réfléchir à la déclaration d'une structure C pour représenter une combinaison de pions. Cette structure devra contenir les informations d'ordre et de couleur des pions. Elle devra également être paramétrable pour s'adapter au nombre de pions fournis en argument de la partie. Cela signifie que votre code doit être réutilisable sans modification pour un MASTERMIND à plus de pions. Cette réutilisabilité doit être intelligemment gérée, i.e., il ne peut pas s'agir d'une constante à aller chercher et modifier dans le code.

Vous devrez également réfléchir à une structure de données pour la gestion de l'historique des combinaisons au cours du jeu. A tout instant de la partie, vous devez pouvoir disposer de l'ensemble des combinaisons qui ont été proposées par l'utilisateur, ainsi que des indications qui leur sont associées. L'ordre de ces combinaisons est également important, et donc doit être pris en compte par cette structure.

Pensez aussi à représenter de manière efficace et pertinente les différentes couleurs.

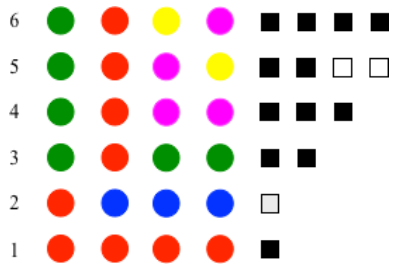


FIGURE 2 – Exemple de résolution par l'ordinateur

3 Les Joueurs

Pour jouer à votre MASTERMIND, il y a deux possibilités :

1. le proposant est l'ordinateur et le devinant est l'utilisateur humain
2. le proposant est l'utilisateur humain et le devinant est l'ordinateur

3.1 Proposant : Ordinateur

Lorsque le proposant est l'ordinateur, celui-çi devra choisir une combinaison de pions de couleurs sans la montrer à l'écran, et l'utilisateur, qui incarnera le devinant, devra la découvrir. Le programme implémenté devra aussi donner le score de l'utilisateur à chacune de ses tentatives.

Le choix de la combinaison, par l'ordinateur, se fera de façon aléatoire à chaque partie. Il faudra alors s'assurer que certaines combinaisons ne sont pas choisies trop régulièrement, à l'aide notamment d'appels bien étudiés à la fonction `srand`, qui permet d'initialiser le générateur pseudo-aléatoire de la librairie standard C. La Sec. 5.1 vous donne de précieuses informations sur comment gérer le pseudo-aléatoire en C.

3.2 Proposant : Être Humain

Lorsque le proposant est l'être humain, l'ordinateur va devoir deviner la bonne combinaison. Il existe une méthode de jeu très simple : jouer au hasard. Cependant, cette méthode n'est pas très efficace. N'ayant que 10 coups pour gagner, l'ordinateur n'a donc que 0.8% de chance de gagner.

L'idée de l'algorithme qui va suivre est d'énumérer les configurations dans un ordre défini à l'avance et d'essayer à chaque fois la première configuration valable.

On muni donc l'ensemble des combinaisons de l'ordre lexicographique (i.e., l'ordre du dictionnaire) :

$$(a, b, c, d) < (e, f, g, h) \text{ ssi } a < e \text{ ou } a = e \text{ et } (b < f \text{ ou } b = f \text{ et } (\ldots))$$

L'idée de l'algorithme de résolution du problème est d'énumérer toutes les configurations possibles dans l'ordre. On commence donc par essayer la configuration $(1, 1, 1, 1)$. [2] Le joueur humain donne alors une réponse (noir, blanc), ce qui élimine un certain nombre de solutions possibles. On énumère ensuite toutes les configurations dans l'ordre en s'arrêtant à la première configuration dont le couple (noir, blanc) corresponde aux valeurs données par l'adversaire.

Un exemple est fourni à la Fig. 2. La première réponse montre qu'il y a une position correcte. La configuration suivante ayant un seul rouge est (rouge, bleu, bleu, bleu). [3] L'être humain répond qu'il y a une boule de mal placée, la configuration suivante devenant (vert, rouge, vert, vert) [4], etc. Au bout de six coups, l'ordinateur a trouvé la solution.

2. 1 représente ici la première couleur, par exemple rouge.
3. Si on considère que le bleu est la deuxième couleur dans l'ordre lexicographique.
4. Si on considère que le vert est la troisième couleur dans l'ordre lexicographique

3.2.1 Génération des Configurations

Supposons qu'on dispose de N couleurs allant de 0 à $N-1$. La première combinaison possible est $(0,0,\ldots,0)$, la suivante est $(0,0,\ldots,1)$ jusqu'à $(0,0,\ldots,N-1)$, puis $(0,0,\ldots,1,0)$, etc. La dernière combinaison est $(N-1,N-1,\ldots,N-1)$.

On peut très facilement générer la position initiale : il suffit de mettre à zéro tous les éléments. Pour générer la position suivante, on incrémente le dernier élément. Si ce dernier élément prend la valeur N, alors on le remet à 0 et on incrémente l'avant dernier. Si cet avant-dernier élément prend la valeur N, on le remet à 0 et on incrémente l'antépénultième, etc. Lorsque le premier élément prend la valeur N, on a fini.

Il faut voir cette génération comme un compteur kilométrique d'une voiture qui affiche les chiffres de 0 à 9.

4 Interface Graphique

Pour créer une interface graphique, GTK propose un système classique à base de *composants*. On dispose du type générique `GtkWidget` pour les désigner. La fenêtre (`GtkWindow`) est le composant dans lequel on place les autres. Certains composants peuvent contenir d'autres composants, on les appelle des *conteneurs* (`GtkWindow` est un conteneur ne contenant pas plus d'un composant, le `GtkBox` est un conteneur permettant de stocker autant d'autres composants que l'on souhaite). On dispose, de plus, de beaucoup de composants classiques pour les interfaces hommes-machines (IHM) : boutons, menus, checkbox, roulettes, zones de dessin, onglets, etc.

Ensuite, on associe à ces composants des *réactions* à certains événements. On pourra ainsi indiquer, par exemple, que la procédure `gtk_main_quit()` sera appelée lorsqu'on clique sur un bouton.

Une fois ces composants créés puis assemblés dans le bon ordre, on peut lancer l'application en donnant la main à GTK (appel de `gtk_main()`). Ce sera GTK qui appelera automatiquement nos réactions selon les événements induits par le comportement de l'utilisateur. On parle de *programmation événementielle*.

4.1 Mise en Page du Mastermind

Pour vous faciliter le travail dans la création de l'interface graphique (sans pour autant brider votre imagination), la Fig. 3 donne une première idée de ce à quoi vous devriez arriver pour votre Mastermind.

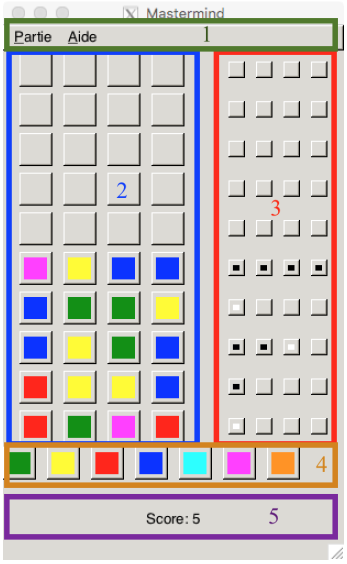


Figure 3 – Exemple d'interface graphique pour le Mastermind.

Figure 4 – Boîte de dialogue indiquant la liste (triée) des 10 meilleurs scores.

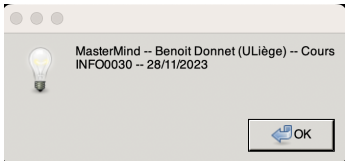


Figure 5 – Boîte de dialogue "A Propos".

Le plateau de votre MASTERMIND est donc composé de cinq zones :

1. La zone verte affichera le menu comprenant (au moins) deux sous-menus :
   (a) *Partie*. Ce menu permet de démarrer une nouvelle partie, de consulter la liste des 10 meilleurs scores (cfr. Fig. 4) ou bien de quitter le jeu.
   (b) *Aide*. Ce menu permet d'obtenir via un item unique, *A propos*, une boîte de dialogue contenant des informations relatives au créateur. Un exemple de boîte de dialogue est donné à la Fig. 5
2. la zone bleue permettra au devinant de faire ses propositions. Il pourrait être intéressant de représenter les propositions par des boutons qui changent de couleur. Si le devinant est l'humain, c'est le clic de souris qui va faire changer la couleur. Sinon, c'est le programme lui-même qui va changer les couleurs en fonction de sa proposition ;
3. la zone rouge affichera le score de chaque proposition ;
4. la zone orange représentera les couleurs à disposition ; [5]
5. la zone mauve indique le score de la partie (i.e., nombre de coups joué). On peut imaginer rajouter des informations dans cette partie, comme un bouton pour redémarrer et/ou quitter le jeu ou encore un bouton permettant de valider une proposition, des boutons permettant de désigner qui a le rôle du devinant et du proposant, etc ;

Une fenêtre "pop-up" indiquera le résultat final (i.e., succès ou échec) pour les joueurs.

4.1.1 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.

4.2 Image et Bouton

Il est possible, en GTK+2, d'afficher une image dans un bouton. Supposons que nous disposions d'une image, `img1.jpg`, et qu'on veut l'afficher dans un bouton `pBouton`. Ceci nécessite trois étapes :

1. charger l'image depuis le disque dur. Eventuellement, on peut redimensionner l'image.
2. créer le bouton.
3. placer l'image dans le bouton

Le bout de code ci-dessous illustre ces trois étapes.

```
GtkButton *charge_image_bouton(){
  //1. Charger l'image et la redimensionner (100*100 pixels)
  GdkPixBuf *pb_temp = gdk_pixbuf_new_from_file("img1.jpg", NULL);
  if(pb_temp==NULL){
    printf("Erreur de chargement de l'image img1.jpg!\n");
    return NULL;
  }
  GdkPixBuf *pb = gdk_pixbuf_scale_simple(pb_temp, 100, 100, GDK_INTERP_NEAREST);

  //2. Créer le bouton
  GtkWidget *pBouton = gtk_button_new();

  //3. Placer l'image
  GtkWidget *image = gtk_image_new_from_pixbuf(pb);
  gtk_button_set_image(GTK_BUTTON(pBouton), image);

  return pBouton;
}//fin charge_image_bouton()
```

5. Des images représentants les différentes couleurs sont mises à votre disposition sur eCampus.

Si on désire remplacer l'image existante d'un bouton, il suffit d'appeler, sur ce bouton, la procédure `gtk_button_set_image` avec la nouvelle image.

Notez qu'à chaque fois que l'on remplace l'image du bouton, on crée un nouvel objet de type `GtkImage`[6] en appelant `gtk_image_new_from_pixbuf`. On pourrait supposer qu'il serait préférable de libérer l'image avant de créer la nouvelle. En réalité, lorsqu'on appelle `gtk_button_set_image`, GTK+2 libère la mémoire occupée par la `GtkImage` précédemment affichée.

5 Éléments Additionnels de C

5.1 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 :

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

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

  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 :

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

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

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

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

```c
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 :

6. A l'instar (par exemple) de `GtkWindow`, un object de type `GtkImage` est aussi de type `GtkWidget`.

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

int main(){
  srand(2);

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

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

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

  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 :

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

6 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. 7).

Le type de SCM à utiliser vous est imposé. Il s'agit de l'instance GitLab de l'Université. Elle est disponible à l'adresse suivante : https://gitlab.uliege.be/. Vous pouvez vous y connecter avec votre compte étudiant.

L'utilisation du GitLab nécessite une inscription préalable avec votre compte étudiant. 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 avec le niveau de visibilité **privé**. Le nom du projet, dans le GitLab, 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. Vous devez **obligatoirement** ajouter les assistants @Emilien.Wansart1 et @Maxime.Goffart1 comme membres du projet avec le rôle "**maintainer**". Une fois cette étape passée, les deux membres du binôme (et uniquement eux) ainsi que les assistants peuvent accéder au GitLab, 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.

7 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, ...). [7]
- 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. Veillez à ajouter le lien vers votre projet sur l'instance GitLab de l'Université (https://gitlab.uliege.be/).
- 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. [8] Le document LaTeXdoit suivre le template donné sur la page web du cours. [9] 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. 8), les sources de votre rapport ainsi qu'une version PDF.

8 Enoncé du Projet

Il vous est demandé d'écrire un programme implémentant le Mastermind telle que décrite dans ce document.

Votre projet devra :

- être soumis dans une archive `tar.gz`, appelée `mastermind_XX.tar.gz`, via la Plateforme de Soumission, où `XX` désigne votre identifiant de groupe. La décompression de votre archive devra fournir tous les fichiers nécessaires dans le répertoire courant où se situe l'archive. Vous penserez à joindre

7. Il n'est pas nécessaire d'indiquer les Invariants de Boucle, les spécifications formelles et, plus généralement, l'approche constructive dans votre rapport.
8. Nous vous renvoyons à votre formation LaTeXreçue en début de quadrimestre.
9. voir eCampus, INFO0030, Sec. Projets/Projet 4: Mastermind

tous les codes sources nécessaires. Votre archive doit être chargée sur la Plateforme de Soumission pour le **Mardi 07/05/2024, 08h00** au plus tard.

- définir les fonctionnalités nécessaires à la mise en place de l'IHM du Mastermind (cfr. Sec. 4).
- 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. [10]
- s'assurer que les structures de données proposées sont implémentées comme des types opaques (quand cela s'avère pertinent).
- 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.
- comprendre un Makefile permettant au moins de

1. compiler votre projet. L'exécution de la commande

```
$>make
```

   permet de générer un fichier binaire, exécutable, appelé `mastermind`.

2. Le fichier binaire généré devra pouvoir être exécuté de la façon suivante :

```
$>./mastermind
```

3. générer la documentation doxygen. L'exécution de la commande

```
$>make doc
```

   devra produire de la documentation au format HTML dans le sous-répertoire `doc/`.

4. générer le PDF de votre rapport. La commande

```
$>make rapport
```

   doit permettre cela.

Si nous appliquons la commande suivante à votre archive :

```
$>tar xvfz mastermind_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, 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. Quant au Makefile, il se trouve à la racine du dossier `GroupeXX`.

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

9 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 [11], 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, ...).

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