Premier bachelier en sciences informatique
Année académique 2018-2019

INFO0030: Mastermind

Groupe 11: Marco Naa, Jordi Hoorelbeke

6 mai 2019

Table des matières

1 Introduction

Dans le cadre du cours INFO0030 : Projet de programmation, il nous a été demandé de réaliser le jeu Mastermind. Ce rapport traitera :

- L'architecture générale du code : les grands concepts et comment ils interagissent entre eux.
- La description des structures de données du code et leur pertinence.
- L'idée derrière certains algorithmes du code. Plus particulièrement,
  - ⋆ L'algorithme qui compare la réponse et la proposition faite pour en retourner un score.
  - ⋆ L'algorithme qui permet à l'ordinateur de faire sa proposition.
- Un profiling du code : le temps d'exécution des algorithmes importants et une estimation de la quantité de mémoire utilisée par les structures de données.
- L'organisation de l'interface graphique.
- L'utilisation du SCM et la coopération au sein du groupe
- Les amélioration possibles au jeu.
- La conclusion : ce que ce projet nous a apporté.

2 Architecture générale du code

Le code suit le pattern MVC. L'utilisateur interragit avec le contrôleur qui lui gère ces interractions et définit le comportement du jeu Mastermind. Le contrôleur implémente le modèle et la vue. La vue quant à elle contient les labels et les images, c'est à dire tous les éléments qui vont être affichés à l'écran, hormis les boutons. La vue implémente le modèle. Pour finir, le modèle contient toutes les fonctionnalités et algorithmes du jeu qui s'occupent des opérations internes.

Trois modes de jeu sont jouables :

- Le mode humain (normal) qui se joue avec les règles classiques du Mastermind.
- Le mode humain (facile) qui est une version simplifée du Mastermind.
- Le mode ordinateur où c'est l'ordinateur qui doit déchiffrer votre combinaison de couleurs.

3 Structures de données

Différentes structures de données ont été implémentées :

- La structure **Ordi** contient tous les outils supplémentaires nécessaires pour l'algorithme de l'ordinateur.
- La structure **Combinaison** représente une combinaison de couleurs et contient donc les informations d'ordre et de couleur des pions. La taille d'une combinaison de couleurs dépend de la macro NB_COULEUR et est donc facilement paramétrable.
- La structure **Historique** contient l'historique des combinaisons réalisées lors d'une partie que ce soit dans le mode humain ou le mode ordinateur.
- La structure **Records** contient les outils suplémentaires nécessaires pour les records.
- La structure **ModeleMastermind**, l'implémentation du modèle du jeu et qui contient, entre autre, les structures Ordi, Combinaison, Historique et Records.
- La structure **VueMastermind**, l'implémentation de la vue du jeu et qui contient la structure ModeleMastermind ainsi que les labels et le chemin vers les images.
- La structure **ControleurMastermind**, l'implémentation du contrôleur du jeu et qui contient les structures VueMastermind et ModeleMastermind ainsi que les boutons.

Le coût en terme de mémoire de ces structures sera abordé dans la section "Profiling et analyse de performance du code".

4 Algorithmes particuliers

Nous avons voulu mettre en avant deux algorithmes, ceux qui sont les pièces maitresses de notre code et qui nous ont pris le plus de temps. Il s'agit de notre algorithme comparant une proposition à la réponse pour fournir le score associé qui figure dans la fonction compare_reponse_proposition() et aussi à celui présent dans la fontion algorithme_ordinateur() qui fournit la proposition de l'ordinateur en fonction des scores des précédentes manches.

4.1 compare_reponse_proposition()

Soit un tableau score de taille NB_PIONS. Cette proçédure compare la réponse cachée et la proposition faite par le devinant et place le score correspondant dans le tableau score. L'algorithme se déroule comme suit :

- Initialisation de deux tableaux copieRep et copieProp de taille NB_PIONS.
- Copiage de la réponse dans copieRep et de la proposition dans copieProp.
- Dans une première boucle, les deux tableaux sont comparés et lorsque dans ces deux tableaux au même indice se trouve la même couleur, les valeurs de ces deux cases sont affectées à zero [1] et la couleur noir est mise dans le tableau score au même indice.
- Dans la deuxième boucle, grâce à une boucle imbriquée qui lit les valeurs du tableau copieProp, on compare les valeurs différentes de zéro de copieProp et celles de copieRep et quand ce sont les mêmes valeurs, un pion blanc est placé à une case égale à 0 dans le tableau score.

4.2 algorithme_ordinateur()

L'algorithme de cette proçédure se déroule selon les règles suiventes :

Soient un tableau temporaire de taille NB_PIONS, couleur la couleur courante, pionNoirs le nombre de pions noirs de la manche courante, pionBlancs le nombre de pions blancs de la manche courante, pion_noir _prec le nombre de pions noirs de la manche précédente et pion_blanc_prec le nombre de pions blancs de la manche précédente.

- A la première manche, une couleur aléatoire est mise dans couleur et à chaque nouvelle manche, une couleur différente de celles déjà utilisées est mise dans couleur.
- Si pionNoirs + pionBlancs est plus grand que pion_noir_prec + pion_blanc_prec :
  Déplacer (pionBlancs-1) fois les cases du tableau contenant une valeur différentes de zéro avec ces trois règles :
  - ⋆ Parcourir le tableau par la gauche.
  - ⋆ Une case contenant une valeur différente de zéro peut être déplacée si et seulement si la valeur de la case à sa droite est zéro.
  - ⋆ Si on arrive à la fin du tableau et que la valeur de (pionBlancs-1) cases n'ont pas encore été déplacées alors recommencer dès le début du tableau.

  Finalement, ajouter la/les nouvelles valeurs à la/les première(s) case(s) égale(s) à zéro à partir de la droite.
- Si pionBlancs + pionNoirs est égal à pion_noir_prec + pion_blanc_prec :
  - ⋆ Même chose que pour le point précédent sauf qu'on décale de pionBlancs fois les cases.
  - ⋆ Décaler la valeur de la case dans la première case valant zéro en allant vers la droite. Si aucune case égale à zéro n'est trouvée avant la fin du tableau, recommencer du début.

1. Mettre la valeur de ces cases à zero permet de ne plus les prendre en compte lors de la prochaine boucle qui compte le nombre de pions blancs et donc d'éviter d'avoir trop de pions blancs

5 Profiling et analyse de performance du code

5.1 Estimation de la quantité de mémoire utilisée par les structures de données

La taille de nos structures est mesurée en octets, Voici la liste des structures et leur taille en considérant que notre constante NB_PIONS valait 4 :

- La structure ModeleMastermind comprend :
  - ⋆ CouleurProposition = **32 octets**
  - ⋆ Couleurscore = **12 octets**
  - ⋆ Combinaison = **34 octets**
  - ⋆ Ordi = **24 octets**
  - ⋆ Historique = **196 octets**
  - ⋆ Records = **52 octets**
  - ⋆ ModeFacile + ModeOrdinateur + Manche + CouleurCourante = **16 octets**
  - ⋆ Tab_imageP = **200 octets**

  Ce qui nous fait un total de **566 octets** pour notre structure ModeleMastermind.
- La structure VueMastermind comprend :
  - ⋆ une structure ModeleMastermind = **566 octets**
  - ⋆ pLabelManche + pLableMode = **16 octets**
  - ⋆ imageP = **50 octets**
  - ⋆ imageS = **50 octets**

  Ceci nous fait alors un total pour Vue de **682 octets**
- La structure ControleurMastermind comprend :
  - ⋆ une structure ModeleMastermind et une structure VueMastermind = **682 octets**
  - ⋆ pBoutonProposition = **320 octets**
  - ⋆ pBoutonScore = **320 octets**
  - ⋆ pBoutonCouleur = **56 octets**
  - ⋆ pBoutonOrdi = **32 octets**
  - ⋆ pBoutonOK = **8 octets**
  - ⋆ nb_boutons_clicked = **2 octets**

Le total pour Controleur et donc pour nos structures est donc de **1420 octets**.

5.2 Temps d'exécution de diverses situations de jeu

Les résultats suivants ont été obtenus grâce à la librairie standard time.h. Avec la fonction clock(), le temps a été récolté au début et à la fin de diverses fonctions et la différence entre ces deux temps a donné les temps d'exécutions. [2]

5.2.1 Dans le mode de jeu humain

- Lorsque l'utilisateur appuie sur le bouton "nouveau" pour recommencer une partie, le temps d'exécution est d'approximativement **0.06 secondes**.
- Lorsque l'utilisateur appuie sur le bouton pour entrer une proposition, le temps d'exécution est d'approximativement **0.002 secondes**
- Lorsque l'utilisateur appuie sur le bouton OK pour valider sa combinaison, le temps d'exécution est d'approximativement **0.003 secondes** dont **0.002 secondes** pour l'algorithme qui se charge de calculer et d'afficher le score.

2. Chaque temps est une moyenne des différents temps récoltés sur plusieurs essais.

5.2.2 Dans le mode de jeu ordinateur

- Lorsque l'utilisateur appuie sur le bouton "nouveau" pour recommencer une partie, le temps d'exécution est d'approximativement **0.05 secondes**.
- Lorsque l'utilisateur appuie sur le bouton OK pour voir la combinaison faite par l'ordinateur et le score associé, le temps d'exécution est d'approximativement **0.14 secondes** dont **0.000009 secondes** pour l'algortihme de l'ordinateur.

6 Interface graphique du jeu

En haut se trouve le menu du jeu Mastermind permettant d'acceder aux différentes options telles que :

- **Partie** :
  - ⋆ **Nouveau** : permet de lancer une nouvelle partie.
  - ⋆ **Records** : permet d'accéder aux dix meilleurs scores.
  - ⋆ **Quitter**
- **Mode** :
  - ⋆ **Proposant : Humain(facile)** : Un mode de jeu facile où l'humain est le proposant et où le score s'affiche dans l'ordre.
  - ⋆ **Proposant : Humain(normal)** : Un mode de jeu normal où l'humain est le proposant et où le score s'affiche dans le désordre.
  - ⋆ **Proposant : Ordinateur** : Un mode où le proposant est l'ordinateur.
- **Aide** :
  - ⋆ **A propos** : Ouvre une fenêtre POPUP contenant les informations sur les auteurs du jeu ainsi que le cours pour lequel il a été fait.
  - ⋆ **Règles du jeu**

En dessous du menu :

- Deux labels vous permettent de savoir dans quel mode et à quelle manche vous êtes et vous donne aussi le score une fois la partie terminée.
- La zone à gauche permet au devinant de faire ses propositions.
- La zone à droite affiche le score de chaque proposition.
- La zone du bas à trois utilitées :
  - ⋆ Sélectionner une couleur.
  - ⋆ Entrer une combinaison de couleurs que l'ordinateur devra déchiffrer.
  - ⋆ Valider sa proposition et/ou passer à la manche suivante en appuyant sur le bouton OK.

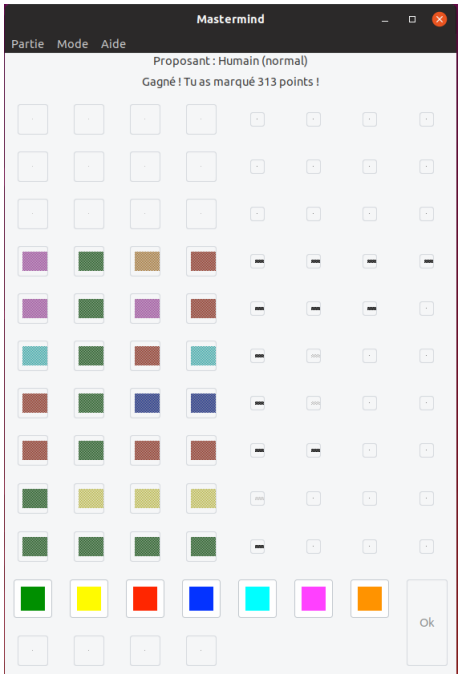

FIGURE 1 – Interface graphique du mode Proposant Humain

FIGURE 2 – Interface graphique du mode Proposant Ordinateur

FIGURE 3 – Boite de dialogue "Records"

FIGURE 4 – Boite de dialogue "Règles du jeu"

FIGURE 5 – Boite de dialogue "A propos"

7 Gestion du code et coopération au sein du groupe

Pour la gestion du code cela s'est passé comme suit :

- Nous avons d'abord discuté de l'architecture générale de notre code puis réalisé un plan afin de mieux organiser notre travail.
- Ensuite, nous avons mis en place les structures et le pattern MVC ce qui nous a servi de base pour notre codage.
- Pour les grands axes du code, notamment l'interface graphique de la fenêtre et les algorithmes complexes tels que les fonctions compare_reponse_proposition() et algorithme_ordinateur(), nous nous sommes retrouvés à l'université afin de pouvoir y réflechir ensembles avant de pouvoir les implémenter.
- Les fonctions "secondaires" telles que toutes les fonctions callback ainsi que le calcul et le tri des scores ont quant à elles pu être codées puis implémentées chacun de notre coté avant d'être mises sur Gitlab.

Le SCM nous à surtout été utile pour fusionner nos codes et pour retracer l'origine d'un problème quand il y en avait. Peut de temps avant la deadline du projet, nous nous sommes rendu compte d'un problème d'accès illégal à la mémoire qui empêchait le jeu de fonctionner. En retrouvant la dernière version du code source qui n'avait pas ce problème et en le comparant avec le code source le plus récent, il nous à été plus facile de localiser l'origine de ce problème.

8 Améliorations possibles du code

On pourrait apporter des améliorations dans notre code, notamment l'algorithme pour que l'ordinateur trouve une combinaison de couleur donnée pourrait être plus performant. Actuellement et selon les divers tests que nous avons pu faire, cet algorithme trouve la solution dans **81% des cas**, et ce après en moyenne **6,2 manches** pour les parties réussies, nous pouvons donc en conclure que cet algorithme est améliorable. En plus de ça, de la même façon que nous avons implémenté un mode facile lorsque le proposant est l'être humain, nous pourrions implémenter un mode difficile où les propositions vides sont acceptées.

9 Conclusion

Pour conlure, nous pouvons dire que ce projet nous a permis de nous rendre compte à quel point il fallait être rigoureux dans l'implémentation du code pour que celui-ci fonctionne. Il nous a aussi appris à faire un plan avant de s'attaquer au codage car dans les grands fichiers il est souvent dur de s'y retrouver lorsqu'on est mal organisé et que faire un bon plan c'est déjà faire une bonne partie du travail, cela nous a aussi aidé à gérer notre temps car cela a été un des grands problèmes. A l'aide d'outils comme Gitlab, Codeshare/Collabedit et Overleaf, nous avons appris à coopérer et à travailler ensembles. Pour terminer, nous avons aussi vu que la pratique était le meilleur moyen de progresser en programmation.