ALGORITHMIQUE ET PROGRAMMATION FONCTIONNELLE - Année 2018/2019

CLASSE PREPARATOIRE CPI2G2

PROJET SEMESTRE 1 :

Mastermind

Made by :
Lilian Naretto, Romain Séré
Ilyes Triki

Table des matières

0.1 Structure du code

Le code du projet **Mastermind** est divisé en 3 fichiers :

— *"Code.ml"* : Module regroupant la plupart des fonctions primaires du projet tels que le nombre de couleurs possibles ou encore la conversion de phrases en listes, etc...

— *"IA.ml"* : Module de l'IA.

— *"ProjetMastermind.ml"* : Fichier principal regroupant les deux modules ainsi que la partie graphique, c'est le pilier centrale du projet.

Dans cette partie du rapport, nous allons explorer en détails les différentes fonctions annexes/-primaires codées durant ce projet.

0.1.1 "Code.ml"

Ce module correspond à la partie 1 du projet, c'est-à-dire le module code. Celui-ci est composé de plusieurs fonctions/types primaires, dont certaines composées de fonctions secondaires :

— *"nombre pions" ; "couleurs possibles" ; "compare"* : Ces trois premières fonctions du projet ont été "oneshot", c'est-à-dire qu'elles étaient simple et qu'elles n'ont pas nécessités des fonctions secondaires.

— *"string of code"* : Cette fonction est composé d'une seule fonction annexe *"string of code2"* qui permet tout simplement à l'aide d'un "match with" d'associer un type pion à son "string".

— *"code of string"* : 3 fonctions composent celle-ci :

  — *"code of string2"* : A l'inverse de *"string of code"*, cette fonction annexe associe un "string" à son type pion grâce à un "match with".

  — *"code of string3"* : Cette fonction permet, à l'aide du module *"str.cma"*, de couper les espaces d'un "string" et de renvoyer plusieurs autres "string".

  — *"code of string4"* : Transforme chaque string d'une liste en type pion à l'aide de la fonction *"code of string2"*.

— *"tous"* : Fonction composé de 2 fonctions secondaires elles mêmes composées de contions tertiaires :

  — *"tous2"* : Elle permet de crée une liste de type couleur à partir d'une liste "d'integer", elle est composé de *"tous3"* qui associe un "integer" à son type pion.

— *"tous4"* : Cette fonction est composé essentiellement de la fonction "ajout" qui est le coeur même de la fonction principale, en effet, elle permet de crée une liste de liste de 4 chiffres représentant les combinaisons possibles.

— *"toutes reponses"* : Composé d'une seule fonction annexe *"ajout2"* permettant de renvoyer les reponses possibles sous forme d'une liste.

— *"reponse"* : Est constitué des fonctions *"noir"* et *"blanc"* permettant de renvoyer la reponse associé à un code donné.

0.1.2 "IA.ml"

Ce module correspond à la partie 2 du projet, c'est-à-dire le module IA. Il est constitué de peu de fonctions mais sa complexité est supérieur à celle des deux autres modules.

— *"nombre methodes"* : Simple "integer" représentant le nombre de methodes d'IA utilisées dans le projet, dans notre cas, 2.

— *"choix"* : Cette fonction est composé d'une fonction secondaire *"listesuprime"* qui prend 2 listes en entré et ressort une seule liste ; permet de supprimer tous les éléments compris dans la première liste, de la deuxième liste (pour l'algorithme de KNUTH), sans compter celles utilisés grâce au module Code.

— *"filtre"* : Composé d'une fonction *"listeS1"* pour la partie de l'algorithme KNUTH.

0.1.3 "ProjetMastermind.ml"

Ce module correspond à la partie 3 du projet, c'est-à-dire le fichier principale regroupant les modules et la partie graphisme. Voici les fonctions qui le constitue :

Pour les fonctions annexes

— *"gagnant"* : Correspond au moment où l'IA ou le joueur gagne, c'est à dire l'affichage du score et les égalités s'il y a.

— *"findugame"* : Fonction associé à la fin de partie, tout simplement.

— *"paire"* : Permet de rendre un "integer" paire.

— *"f"* et *"f2"* : Associe sa "Some" à une liste/un duet.

— *"demandecode"* : Demande à l'utilisateur d'entrer un code jusqu'à ce que celui-ci ai une syntaxe correcte et va renvoyer ce code.

— *"lirerep"* : Demande d'entrer la réponse au code de l'IA.

— *"affichageplat"* : Affiche le plateau du joueur (codes proposés et réponses associées) lors des parties où le joueur doit deviner le code de l'IA.

Pour les fonctions principales

— *affichage* : Affiche les différentes étapes dans les parties du jeu où le joueur doit deviner le code de l'IA.

— *"affichageIA"* : Afficher les différentes étapes du jeu dans les parties où l'IA doit deviner le code du joueur.

— *"humainvsIA"* : Lance la partie où l'IA doit deviner le code de l'humain.

— *"IAvshumain"* : Lance la partie où l'humain doit deviner le code de l'IA.

— *"num"* : Fait le jeu en alternant le lancement de la fonction humainvsIA et iavsHUMAIN.

— *"mastermind"* : Récapitule les choix du joueur et lance le jeu.

— *"start"* : Demande au joueur plusieurs paramètres et lance le jeu grâce à la fonction mastermind.

0.2 Choix algorithmiques

Le choix algorithmique du projet était assez simple, chaque fonctions (dû moins, les plus compliqués) étaient divisées en plusieurs autres fonctions, pour ainsi permettre une meilleur manipulation du code (il vaut mieux un code composé de 5 autres qu'un seul bloc de code).
Pour ce qui est du choix des fichiers, le raisonnement nous paraissait évident, 1 fichier pour une partie, c'est à dire *"IA.ml"* ,*"projetmastermind.ml"* et *"Code.ml"*.

La partie graphisme
Pour le choix du graphisme du projet, nous avons opté pour un aspect simple mais quand même agréable à regarder. L'absence de cours pour la partie graphisme du projet était assez handicapant, du coup nous avons dû apprendre de nous même le graphisme en ocaml, ce qui a conclu en une interface basique, c'est-à-dire avec des "string", mais tout de même agréable, avec des couleurs, des animations et un grand titre "MASTERMIND" en couleur au debut du jeu.
La contrainte du graphisme était la place en ligne de code que cela prennait, ce qui entrainait donc sur des difficultés d'optimisation du code, que nous allons parler dans la sous partie suivante.

0.2.1 Les diffcultés rencontrées

Tout au long du projet, nous avons rencontrer plusieurs difficultés, que ce soit des erreurs de compilations, majoritairement, mais aussi d'optimisations.

La compilation
Nous avons rencontrées plusieurs erreurs de compilations, notamment à cause de l'utilisation du module "str.cma" qui nous a facilité la tâche pour certaine fonctions. Il faut aussi prendre en compte que c'était la première fois que nous utilisions les modules sous ocaml, notre methode de codage en était influencer, de par le fait que nous étions en groupe, mais qu'il fallait aussi partager notre projet sous un git.

KNUTH
Cette IA doit bien être le code le plus difficile et le plus optimisé qu'on ai jamais écrit en ocaml, il faut savoir que le code d'origine prennait environ 50 lignes de code que nous avons réussi à ultra optimisé en quelques lignes.
Au moins la moitié du temps se composait de la compréhension de ce qu'était un algorithme de KNUTH, aucun site internet n'arrivait à nous faire comprendre son, fonctionnement, nous avons dû fouiller dans plusieurs sites, notemment anglais, puis regrouper nos informations pour enfin comprendre la façon dont nous allions le coder.

Le graphisme
Les plus gros problèmes se trouvait ici, nous n'avions aucune idée de comment créer une interface graphique sous ocaml, nous avons dû faire des recherches, regarder des videos, pour enfin comprendre plus ou moins son fonctionnement. La majorité du temps, les problèmes résiliaient dans la finition de l'affichage, comme par exmeple le titre au debut du jeu qui nous voulait JAMAIS s'afficher de manière voulu ou encore d'autres erreurs d'affichages comme des loadings screens infinis.

0.3 Répartition du travail

Dans cette partie nous allons aborder le sujet de la répartition du travail au sein du groupe, il faut savoir qu'au debut, il n'y avait pas vraiment de répartition, chacun codait la fonction dont il avait envie (pour le fichier *"Code.ml"* par exemple), sauf que par la suite, nous avons réalisé l'importance d'une organisation, qui s'est effectué pour la deuxième moitié de la première partie et le reste du projet.

0.3.1 Répartition simplifié

Voici une répartition simplifié des tâches au sein du groupe :

	Romain	Ilyes	Lilian
"Code.ml"	30%	30%	40%
"IA.ml"	50%	20%	30%
"projetmastermind.ml"	50%	30%	20%
Git et LaTeX	20%	30%	50%

La répartition sous forme de pourcentage est approximatif, elle est juste la pour indiquer de façon très bref la contribution au projet de chacun.

0.4 Bilan

En conclusion, nous pouvons dire que le ocaml n'est vraiment pas un langage que nous adorons, loin de la , il est complexe, peu compréhensible mais aussi très peu fournis en information sur internet.
Pour ce qui est du projet, nous y avons vraiment passer beaucoup de temps comme cela peut se voir par le projet final (on l'espère), nous avons vraiment soigner les détails, comme le graphisme, avec l'intégration de couleurs mais aussi l'optimisation du code, le seul regret pourrait être la lenteur de l'IA KNUTH, mais cela dépend seulement de l'ordinateur où le programme est lancé (15 secondes sur un ordinateur de l'EISTI, 1 seconde sur le pc de Lilian), mais sa lenteur est complété par son éfficacité, en effet, l'IA KNUTH finit une partie en moyenne en 5 tours.
Au final, ous avons aimé ce projet, même si l'ocaml, c'est vraiment pas fait pour nous.