
Ecole Internationale des Sciences du Traitement de l'Information

CaML

---

# Projet Mastermind

---

*Auteurs :*
M. Valentin LABAT
M. Mathieu SURMAN
M. Hugo BERNEDO

*Encadrant :*
Pr. Romain DUJOL

Version 1.0 du
18 janvier 2019

# Table des matières

# Introduction

Le Mastermind ou Master Mind est un jeu de société pour deux joueurs dont le but est de trouver un code. C'est un jeu de réflexion, et de déduction, inventé par Mordecai Meirowitz dans les années 1970 alors qu'il travaillait comme expert en télécommunications.

Il se présente généralement sous la forme d'un plateau perforé de 10 rangées de quatre trous pouvant accueillir des pions de couleurs. Le nombre de pions de couleurs différentes est de 6 et les couleurs sont dans la version originale2 : jaune, bleu, rouge, vert, blanc, noir. Il y a également des pions noirs et blancs utilisés pour donner des indications à chaque étape du jeu.Il existe de nombreuses variantes suivant le nombre de couleurs, de rangées ou de trous et les couleurs utilisés pour les pions. Le jeu peut par exemple contenir 8 couleurs (rouge, jaune, bleu, orange, vert, blanc, violet, rose) et les pions indicatifs peuvent être jaunes et rouges. Il y a aussi une version qui propose de découvrir un code de 5 couleurs en 12 rangées : le "Super" Master Mind.
Un joueur commence par placer son choix de pions sans qu'ils soient vus de l'autre joueur à l'arrière d'un cache qui les masquera à la vue de celui-ci jusqu'à la fin de la manche. Le joueur qui n'a pas sélectionné les pions doit trouver quels sont les quatre pions, c'est-à-dire leurs couleurs et positions. Pour cela, à chaque tour, le joueur doit se servir de pions pour remplir une rangée selon l'idée qu'il se fait des pions dissimulés. Une fois les pions placés, l'autre joueur indique :

- le nombre de pions de la bonne couleur bien placés en utilisant le même nombre de pions noirs ;
- le nombre de pions de la bonne couleur, mais mal placés, avec les pions blancs.

Il arrive donc surtout en début de partie qu'il ne fasse rien concrètement et qu'il n'ait à dire qu'aucun pion ne correspond, en couleur ou en couleur et position. La tactique du joueur actif consiste à sélectionner en fonction des coups précédents, couleurs et positions, de manière à obtenir le maximum d'informations de la réponse du partenaire puisque le nombre de propositions est limité par le nombre de rangées de trous du jeu. Dans la plupart des cas, il s'efforce de se rapprocher le plus possible de la solution, compte tenu des réponses précédentes, mais il peut aussi former une combinaison dans le seul but de vérifier une partie des conclusions des coups précédents et de faire en conséquence la proposition la plus propice à la déduction d'une nouvelle information.Le joueur gagne cette manche s'il donne la bonne combinaison de pions sur la dernière rangée ou avant. Dans tous les cas, c'est à son tour de choisir les pions à découvrir.

*Source : Wikipedia.org*

# 1 Composition du programme

Nous avons choisi de décomposer le projet en plusieurs fichiers, ainsi il sera composé de :

- ✓ Code.ml
- ✓ IA.ml
- ✓ inter.ml
- ✓ knuth.ml
- ✓ Menu.ml
- ✓ AlgoNaif.ml

Le fichier *Code.ml* comprends le corps du programme c'est à dire toutes les fonctions nécessaire à son bon fonctionnement.

Le fichier *IA.ml* lui comprends le code source de notre intelligence artificielle.

Le fichier *inter.ml* lui sera composé du code source d'une interface 2D qui fait apparaître une fenêtre composée des pions déjà testé.

Le fichier *knuth.ml* lui sera composé de deux algorithme de recherche dans un premier temps celui de Knuth-Morris-Pratt puis celui de Boyer-Moore.

Enfin le fichier *Menu.ml* sera le corps du code source nécessaire lors du déroulement du jeux.

# 2 Explications des codes

## 2.1 Code.ml

Le ***type pion*** est la modélisation des pions du jeux, nous avons fait le choix de les modéliser par des entiers.

La fonction ***nombre de pions*** permet d'initialiser le jeux sur le nombre de pions à trouver.

La fonction ***couleurs possibles*** permet de donner les conditions sur le nombre de couleurs du jeu.

La fonction ***compare*** prends en paramètre deux codes, il sera comparé il en resortira 0 si les codes sont identiques sinon il en resortira un entier positif si le premier code est plus grand que le second sinon un entier négatif.

La fonction ***ecriture base b*** converti une écriture en une écriture dans une autre base.

La fonction ***tous les codes*** fait la liste de tous les codes permis.

La fonction ***string of code*** fait la conversion d'un code vers une chaine de caractère dans l'utilité de servir à l'affichage.

La fonction ***match ASCII int*** a partir d'un caractère elle renvoie son numéro ASCII.

La fonction ***code of string1*** renvoie le code à partir d'une chaîne de caractères.

La fonction ***verification*** verifie si le code convertie appartient a la liste de tous les codes possibles.

La fonction ***code of string*** convertit la chaîne de caractère en code.

La fonction ***arbitrages sachant bp*** gènère un code sachant les bien placés.

La fonction ***toutes reponses*** permet de faire la liste de toutes les réponses possibles.

La fonction ***bienplaces*** calcule les pions bien placé.

La fonction ***compter et retirer bp*** construit un triplet avec le nombre de pion bien placé le code secret le moins bien placé l'essais moins les bien placés.

La fonction ***occ*** compte les occurrences.

La fonction ***compter mp*** compte les pions qui sont bien placés dans les deux listes .

La fonction ***reponse*** calcul la réponse d'un code par rapport au code caché ainsi elle retourne un couple composé du nombre de pions bien placés et du nombre de pions mal placés et None si la réponse ne peut pas être calculée.

## 2.2 IA.ml

La fonction ***nombre de méthodes*** permet de rentrer le nombre de méthodes implémentés.

La fonction ***choix*** permet à l'utilisateur de choisir l'intelligence artificielle.

La fonction ***filtre*** filtre les codes possibles puis retourne la liste des codes possibles.

## 2.3 inter.ml

Le type ***relief*** permet de creer un effet de perspective.

Le type ***box config*** permet de creer les paramètres de la case que l'on va utiliser.

Les fonctions ***draw*** sont les fonctions qui vont permettre le dessin de notre interface.

Les fonctions ***boder*** permette de paramètrer les limites du tableau.

La fonction ***erase box*** efface les boites.

Le type ***position*** va permettre de choisir la position des éléments.

La fonction ***draw string in box*** nous permet d'afficher du texte dans les boites de l'interface.

La fonction ***set gray*** initialise la couleur grise de notre éléments.

La fonction ***create grid*** permet une création de la grille qui contient les boites.

## 2.4 knuth.ml

La fonction ***compare substrings*** compare deux sous chaînes de caractères afin de verifier si : $w_k...w_{k+s-1} = w'_{k'}...w'_{k'+s-1}$ sont les mêmes le résultat sera booléen.

La fonction ***occurences*** prends deux chaînes de caractères pour retourner la liste des occurences d'un mot x dans un texte t.

La fonction ***border length*** calcule la longueur du bord d'un mot. Cette fonction permet de connaître la longueur d'un bord de chaque préfixe de x c'est à dire tous les mots de la forme : $x_0...x_{s-1}$

La fonction ***border*** prends en argument un mot x et retourne un tableau bêta contenant les valeurs de $\beta(j)$ pour j allant de 0 à m.

La fonction ***kmp*** est la fonction d'imbrication de tous l'algorithme de Knuth-Morris-Pratt il renvoie une liste d'entier qui sera composé de la position de la sous chaîne que l'on cherche si elle existe et sinon elle renverra une liste vide.

La fonction ***reskmp*** permet de dire à l'utilisateur si le code rentré est le bon ou non.

La fonction ***first occurence*** calcule la position de la première occurrence de chaque lettre dans un mot x. Son résultat sera sous forme de tableau.

La fonction ***good suffix*** utilise la fonction borders et prends en argument un mot x et retourne le tableau des valeurs $\gamma(j)$.

La fonction ***bm*** utilise les fonctions first occurence et good suffix et elle est la fonction de l'algorithme de recherche *Boyer et Moore*.

## 2.5 Menu.ml

La fonction ***option to thing*** renvoie ce qu'il y a dans un type option.

La fonction ***choisir code*** demande à l'utilisateur de rentrer le code qu'il veut faire deviner et vérifie si le code entré est valable au vue des paramètres du jeu si il ne l'est pas il renvoie un message d'erreur et demande à l'utilisateur de recommencer la saisie.

La fonction ***choisir nbr pions*** initialise les paramètres du jeux en demandant à l'utilisateur de rentrer le nombre de pions souhaités pour jouer. Si le nombre de pions et incorrectes un message d'erreur apparaît il faut alors ressaisir le nombre de pions souhaité.

La fonction ***choisir nbr couleurs*** initialise les paramètres du jeux en demandant à l'utilisateur de rentrer le nombre de couleurs souhaités pour jouer. Si le nombre de couleurs et incorrectes un message d'erreur apparaît il faut alors ressaisir le nombre de couleurs souhaité.

La fonction ***choisir mode*** demande à l'utilisateur de choisir le mode de jeux qu'il souhaite *"Mode de jeu IA vs IA"* ou *"Mode de jeu IA vs Joueur"* ou *"Mode de jeu J vs J"* si la saisie est incorrecte un message d'erreur apparaît et demande de ressaisir les bonnes informations.

La fonction ***jeu*** est la fonction qui rassemble tout le code pour que le jeu fonctionne.

## 2.6 AlgoNaif.ml

La fonction ***filtrerec*** est une fonction récursive qui permet de filtrer la liste des codes afin d'en ressortir une nouvelle liste de code possible.

La fonction ***eval*** permet d'évaluer le nombre de bonne réponse du joueur.

La fonction ***score*** permet d'attribuer un score à un code.

La fonction ***meilleur score*** retourne la liste des codes qui ont le meilleur score.

# 3 Répartition du travail

| Mathieu SURMAN | Valentin LABAT | Hugo BERNEDO |
|---|---|---|
| AlgoNaif.ml<br>(fct filtrerec ; eval ;<br>score ; meilleur score) | Menu.ml<br>(toutes fcts) | AlgoNaif.ml<br>(fonction AlgoNaif) |
| Code.ml<br>(fct ecriture tous les codes ; verification ;<br>base b ; tous ;<br>arbitrages sachant bp ; toutes reponses ;<br>bien places ; occ ;<br>compter mp ; reponse ;<br>tous les codes ; verification ;<br>compter et retirer.) | Code.ml<br>(fct compare ;<br>match ASCII int ;<br>code of string1 ;<br>string of code ;<br>code of string). | knuth.ml<br>(Rechercher des algorithme de Knuth-Morris-Pratt et de Boyer-Moore avec implémentation des fonctions). |
| IA.ml<br>(fct choix ; filtre). | README.ml<br>(Ecriture du readme). | Inter.ml<br>(Rechercher de la documentation sur les interfaces 2D et implémenter les fonctions). |
| | | Rapport du projet<br>(Rédaction intégrale du rapport). |

# Bilan du projet

Ce projet fut enrichissant, effectivement chacune des personnes du groupe à su trouver sa place. La répartition du travail s'est faite naturellement suivant les compétences des personnes.
Le projet nous a permis de mettre en application nos connaissances du langage CaML ainsi nous avons pu réaliser la modélisation d'un jeu de société : Mastermind. La mise en place de l'interface graphique s'est avérée complexe ainsi nous n'avons pu la finaliser.
L'algorithme de Knuth-Morris-Pratt est fonctionnel ainsi que celui de Boyer-Moore mais nous n'avons pas réussi à découper les étapes de ces algorithmes afin qu'il soit fonctionnel pour une utilité tour pas tour.
Le projet nous a fait acquérir une rigueur de rédaction, notamment la partir commentaire et explication des fonctions.
Nous avons essayer de faire une interface avec l'utilisateur la plus intuitive possible, avec des explications lors de chaque étape afin faciliter l'utilisation du jeu.