Mastermind en `Ocaml`

Rapport projet par Paul Planchon et Thomas Durand

I. Introduction

Voilà après plus d'1 mois et demi de travail notre Mastermind terminé. Le travail fut long mais nous sommes plutôt fière du résultat et nous espérons que vous prendrez du plaisir à y jouer.

Mais tout d'abord quelques précision technique. Notre partie graphique a été réalisée à l'aide de la SDL c'est pour cela que pour compiler notre code il vous faudra au préalable installer la SDL (Simple DirectMedia Layer) pour Ocaml (il vous suffit de suivre les instructions du volet Installation de `https://vog.github.io/ocamlsdl-tutorial/`)

Ensuite la compilation se fait à l'aide d'un MakeFile pour simplifier l'utilisation des modules et des bibliothèques, c'est pour cela qu'il vous faudra utiliser la commande `make` dans un terminal pour tout compiler d'un coup. En enfin l'exécution se fait grace a la commande

Code Source bash - 1: Lancer l'application

```
1 ./game nombre_de_pion nombre_de_couleurs
```

le nombre de pion doit etre compris entre 2 et infini mais attention au débordement d'écran et le nombre de couleur entre 2 et 11 mais attention si il y a trop de couleurs les calculs seront très gourmand. Bonne partie ;)

II. Les Intelligences artificielles

II.1. IA de Knuth

L'algorithme de Knuth est le plus connu des algorithme permettant de résoudre un Mastermind en un minimum de coup, mais sa compréhension n'est pas la plus simple. Il est le fruit du travaille de Donald Knuth un mathématicien et informaticien Américain ayant beaucoup travaillé sur l'algorithmie et très connu pour être l'inventeur de Tex, un éditeur de texte OpenSource.

Tout d'abord pour connaître et comprendre cet algorithme nous avons fais appel a Internet et grâce a la page Wikipédia américain du Mastermind `en.wikipedia.org/wiki/Mastermind_(board_game)` qui a été notre principal source de documentation car le principe de déroulement de Knuth y est expliqué étape par étape.
Pour faire simple, le fonctionnement est divisé en deux partie : Le filtre et le choix du Code à jouer. Le filtre est la pour éliminer les codes qui ne peuvent pas être le code secret en se basant sur le score obtenue lors des essaies précédents. Pour cela chacun des codes potentiels et tester avec le code qui vient d'être joué, si le score obtenue est différent du code obtenue par rapport au code secret alors le code ne peut pas être le code secret et il est retiré des codes potentiel.

Une fois l'opération de filtrage effectué il faut choisir le prochain code à jouer. Pour ce faire plusieurs méthodes s'offre a nous. On peut tout d'abord choisir le prochain code aléatoirement parmi les codes potentiels, mais cette opération n'est pas la plus efficace. En effet le plus optimisé serait de jouer le code qui serait le plus probable, c'est à dire celui qui éliminerai le plus de code des codes probable . Pour ce faire l'algorithme du MinMax est le plus adapter et grâce a ce l'ordinateur joue le code le plus adapter. Mais

ces techniques ont leurs limites en effet bien qu'elles trouvent a chaque fois en moins de 5 coups pour des rangé de 4 pions, les calculs sont très gourmands et des lors que les paramètres sont de 5 ou 6 pions avec 7 ou 8 couleurs les calculs deviennent interminables.

II.2. IA Génétique

La différence entre l'IA de Knuth et l'IA génétique, c'est que celle génétique est bien moins documentée. Pour faire l'IA avec laquelle vous pouvez jouer nous avons du lire plusieurs papier universitaires en anglais, et y décortiquer tous leurs détails.

Nous n'avons pas réussis à le faire entièrement, car de temps en temps, l'algorithme tourne sur des listes très très grande, ce qui n'est pas normal. Cette grande liste entraine un très grand temps d'éxécution. Nous n'avons pas réussi à trouver la cause de ce problème. Nous allons maintenant vous présenter cet algorithme. Un algorithme génétique est un algorithme basé sur le principe d'évolution de Darwin. Chaque population a des parents, qui vont muter : des crossover, des permutation et des mutations vont les aider à changer.

Cette nouvelle population est alors triée via une fonction propre à l'algorithme, la fonction `fitness`. Cette fonction fitness va trier la nouvelle population et créer les codes qui seront aptent à être joués. L'équation de la fonction fitness que nous avons utilisé est la suivante :

$$f(c;i) = \sum_{q=i}^{i} |X'_q(c) - X_q| + \sum_{q=1}^{i} |Y'_q(c) - Y_q| \tag{1}$$

Cette équation parrait compliquée, mais une fois comprise elle est simple : c'est la somme des différence entre reponse du à tester sur les anciens codes. Dans les papiers, cette fonction peut-être pondérée, mais la plupart de ces même papier ne le conseillent. A chaque code est attribué sa valeur fitness, les codes ayant la valeur fitness la plus faible sont ceux utilisées. On recommence la procédure génération des fils (avec l'évolution) et attribution de la fonction fitness tant que le nombre génération ne depasse pas une constante fixée ou que la taille des codes a tester atteind une autre constante.
Une fois les codes générés, le prochain code à jouer les est le premier code non joués dans ceux générés.

Ce code est très intéressant car c'est une méthode un peu plus organique de réponse à un problème très mathématique. Si jamais, vous voyez que notre interface graphique est figée, relancez le programme. Vous êtes dans le cas des milliers de code à tester...

III. Le module graphique

Pour notre projet une partie Graphique était fortement recommandée et donc indispensable a nos yeux. Nous avons donc décidé de nous y atteler des les premiers semaines du projet afin de prendre de l'avance sur notre planning et de pouvoir tester nos algorithmes dans de bonne conditions.

Dans le sujet proposé l'utilisation du module Graphics était recommandé et c'est ce que nous avons utilisé. La documentation Ocaml étant bien faites nous n'avons pas eu beaucoup de mal a realiser notre premier plateau de jeu en un temps relativement coups (environs 1 semaine). Mais très vite des barrières se sont présentées à nous : le module Graphics est très très limité. La gestion des events est catastrophique et des tas de fonctions elementaires manque a ce module, par exemple l'ajout d'images et même des choses basiques comme augmenter la taille de la police. Toutes ces raisons ont fait que la programmation était longue et laborieuse pour un résultat qui certes fonctionné mais ne nous correspondais pas.

C'est pourquoi nous avons utilisé la SDL, un wrapper de la SDL disponible sous le nom de `ocamlsdl`. C'est un bon wrapper de la SDL C. Il nous a permis, avec un peu de recherche sur Internet de faire l'interface graphique que vous voyez aujourd'hui. L'utilisation de cette librairie est très similiare à la SDL C ou C++.

IV. Ce que nous a apporté le projet

Ce Mastermind est notre premier gros projet dans un langage fonctionnel. Bien que nous ayons déjà bien pratiqué le langage Ocaml a travers des exercices et des TP en cours l'expérience reste quand même très différente. En effet la façon de procéder par rapport a un langage procédural est très différente et il nous a fallut un bon moment avant d'arriver a penser nos fonction comme un tous en fonctionnel. Mais le résultat est la et la puissance aussi car de gros calculs comme celui de l'algorithme de Knuth s'exécute dans des temps records.

En ce qui concerne l'utilisation de Git la première expérience est plus que convaincante. Précédemment nous utilisions des services comme Google Drive pour nous permettre de travailler a distance mais je ne compte plus le nombre de version inutile qui traîner dans des dossier avec des noms incompressible. Avec Git tous c'est fais très naturellement après un petit apprentissage des commandes basiques sans aucun regret.

V. Conclusion

Pour conclure ce projet n'as clairement pas été notre préféré entre toutes les années de CPI. En effet nous l'avons trouver étonnamment simple et surtout beaucoup plus simple que le précédent (le Qwirkle). En effet les règles du Mastermind n'étant pas très compliqué elles ont été assez rapide a transcrire en OCaml.

De plus l'algorithme de Knuth bien qu'étant très complexe était déjà entièrement expliqué sur internet il nous été également très « facile » de le comprendre et de l'implémenter (par rapport au Qwirkle ou aucun algorithme ne nous été proposé). La difficulté résidé donc dans le fait que le langage utilisé est un langage fonctionnel mais nous avions eu tout le temps de maîtriser la plus part des difficulté grâce au TD vu en cours. C'est également pour rechercher cette difficulté que nous avons décide d'utiliser la SDL avec succés.

Mais malgré ces petits défauts nous avons quand même pris du plaisir comme a chaque fois dans les projet d'informatique et nous réutiliserons le OCaml avec joie.