Projet OCaml
Le jeu du Mastermind

1 Introduction

Le but de ce projet est l'écriture d'un programme permettant de jouer au Mastermind. Il s'agira pour l'ordinateur de découvrir une combinaison de couleurs tenue secrète par le joueur humain, à travers un protocole de questions / réponses.

2 Description du jeu

Le jeu du Mastermind (légèrement revisité ici) se joue à 2 joueurs. Le joueur A, qui dispose au départ d'autant de pions de C couleurs différentes que nécessaire, choisit une combinaison secrète de K pions, pas nécessairement de couleurs différentes, pour K compris dans un intervalle $[M, N]$.

Le joueur B connaît les C couleurs ainsi que les valeurs de M et N (mais pas K). Il a pour objectif de deviner la combinaison secrète du joueur A en proposant successivement des combinaisons de pions. Le joueur A doit alors indiquer pour chaque essai le nombre de pions bien placés (i.e. de la bonne couleur et à la bonne position), ainsi que le nombre de pions mal placés (i.e. de la bonne couleur mais à la mauvaise position).

Dans le jeu original, si le joueur B trouve la bonne combinaison avant un nombre de coups maximum, il est déclaré vainqueur, sinon le joueur A a gagné. Ici, nous permettons au joueur B de proposer des combinaisons jusqu'à ce qu'il trouve la bonne.

3 Hypothèses de travail

L'objectif est d'obtenir une application **principale** OCaml qui puisse se compiler et s'exécuter sans le recours de l'interprète `ocaml`. Vous tiendrez le rôle du joueur A, et votre application principale celui du joueur B. Vous devrez laisser jouer l'ordinateur jusqu'à ce qu'il trouve votre combinaison cachée, après quoi votre programme devra simplement s'arrêter.

Votre programme doit pouvoir trouver **systématiquement** la combinaison cachée du joueur en un nombre de coups fini et pas trop grand (typiquement une dizaine de coups pour $N = 5$ et $C = 8$). L'utilisation de méthodes purement aléatoires est donc exclue. Votre programme devra donc exploiter **au maximum** chacune des réponses faites à ses essais successifs, de telle façon que l'ensemble des combinaisons jouables par votre programme s'amenuise après chaque essai.

Une conséquence de ce fonctionnement est que votre application devra pouvoir déterminer si vos réponses successives sont toujours consistantes, i.e. qu'il y a toujours une combinaison possible non essayée. Il devra donc être impossible de tricher contre votre programme sans que celui-ci ne s'en aperçoive finalement.

3.1 Dimensionnement

Le nombre de couleurs disponibles sera fixé à $C = 8$, correspondant à l'ensemble $\{Rouge, Vert, Jaune, Bleu, Magenta, Blanc, Noir, Cyan\}$. Votre application principale devra fonctionner de manière raisonnable en temps et en mémoire pour des combinaisons de taille $N = 6$.

3.2 Modules prédéfinis

Les pions colorés utilisés dans le jeu de Mastermind sont définis dans un module `Pion`. Ils peuvent être effectivement affichés en couleur dans tout terminal, ainsi que dans l'interprète OCaml grâce au script `script_pion.ml`.

Pour le dialogue de votre application principale avec le joueur A, rôle tenu par l'utilisateur, vous devrez utiliser **exclusivement** les fonctions du module `Io_mastermind` fourni. Le respect de ces contraintes est nécessaire afin que des procédures de tests automatiques puissent être mises en place.

Les tests seront en effet effectués avec une autre implantation de ce module qui tiendra le rôle du joueur A et permettra de jouer automatiquement des parties sans intervention de l'utilisateur.

3.3 Module principal à implanter

Le fichier interface `mastermind_principal.mli` contient les différentes déclarations de type et de fonctions à implanter. Vous êtes bien sûr libre de définir plus de types et de fonctions que ce qui apparaît dans l'interface, dans la mesure où vous respectez et ne modifiez pas celle-ci.

Il vous est demandé d'utiliser **obligatoirement** une représentation arborescente (par exemple un arbre de type n-aire) pour représenter l'ensemble des combinaisons possibles à tout instant de la partie, i.e. le type `combinaisons`. Vous êtes libre de choisir tout type de structure appropriée, mais il est important que vous réfléchissiez au meilleur choix avant de vous lancer dans le codage des fonctions.

Ce choix, par rapport à une solution plus naïve où l'on manipulerait la liste de toutes les combinaisons par exemple, doit permettre à votre algorithme de fonctionner avec un temps de réponse et une consommation mémoire acceptable.

Enfin, un critère raisonnable pour déterminer la fin de la partie consiste à vérifier que le nombre de pions bien placés est égal à la taille de la combinaison proposée par le joueur B. Or, la taille des combinaisons n'étant pas fixée, ce critère n'est valide que lorsque le joueur B propose une combinaison parmi les plus longues possibles. Vous veillerez donc à respecter cette condition.

3.4 Fournitures

Vous disposerez du module `Io_mastermind`, du module `Pion`, du fichier script `script_pion.ml`, de l'interface du module `Mastermind_principal`, ainsi que du fichier `README` expliquant le rôle des différents fichiers, dans une archive `mastermind_etud.tar`, disponible sur la page des TP OCaml.

Le contenu de l'archive s'extrait grâce à la commande suivante :

```
tar xvf mastermind_etud.tar
```

4 Exemple de partie détaillée

On dispose dans cet exemple, pour des raisons de concision, de $C = 3$ couleurs $\{R, V, B\}$ (pour $Rouge$, $Vert$ et $Bleu$), et on cherche des combinaisons de K pions, pour $K \in [2, 3]$.

Joueur	Action	Combinaisons possibles restantes
A	Choix de BRR	RR RV RB VR VV VB BR BV BB RRR RRV RRB RVR RVV RVB RBR RBV RBV VRR VRV VRB VVR VVV VVB VBR VBV VBB BRR BRV BRB BVR BVV BVB BBR BBV BBB

B	Essai de RRR	RR RRV RRB RVR RBR VRR BRR
A	2 bien placés 0 mal placé	

B	Essai de RRV	RBR BRR
A	1 bien placé 1 mal placé	

B	Essai de RBR	BRR
A	1 bien placé 2 mal placés	

B	Essai de BRR	BRR
A	3 bien placés 0 mal placé	

B	Gagné !

5 Travail demandé

Ce projet sera réalisé en **monôme**, à l'exclusion de toute forme de partage, copie, réutilisation de code entre étudiants. À l'issue de ce projet, vous devrez envoyer par courrier électronique à votre enseignant de TD un fichier archive `«votre nom».tar` contenant :
– un rapport résumant le travail effectué ;
– l'ensemble des sources OCaml des modules implantés.

Ce travail sera testé pendant 10 minutes environ par un enseignant, en votre présence. À l'issue de cette séance de test, vous rendrez également une version imprimée du rapport.

5.1 Programmation

Vous devrez programmer dans un style fonctionnel, i.e. sans références, sans tableaux et sans procédures. Vous êtes encouragés à utiliser les fonctions prédéfinies, notamment celles du module `List`, en particulier les itérateurs.

Le choix des algorithmes n'interviendra pas dans la notation, dès lors que les solutions adoptées sont justifiées et commentées. De plus, ces solutions devront être prioritairement motivées par une exigence de clarté, d'exhaustivité et de concision. Les fichiers OCaml rendus devront impérativement pouvoir être compilés, afin de produire une application principale indépendante de l'interprète OCaml.

Lorsque votre programme **semblera** fonctionner, il est important que vous le soumettiez à des tests significatifs, inclus dans le rapport, qui doivent en montrer le bon fonctionnement dans tous les cas que vous jugerez utile de distinguer. Vous ferez apparaître les tests pour chaque fonction individuellement (tests unitaires) ainsi que les tests de l'application principale (tests d'intégration).

Le texte source, inclus également dans le rapport, devra bien sûr être clair, lisible, correctement indenté, intelligemment annoté, en respectant les règles évoquées en TD/TP.

5.2 Rapport et Tests

Le travail de conception et de programmation effectué devra figurer *in extenso* dans le rapport de projet à rendre. Vous devez clairement détailler les découpages en fonctions et les algorithmes de votre programme, expliciter et **justifier** les choix de conception. Le principe est d'expliquer suffisamment clairement les algorithmes de manière à pouvoir comprendre le programme sans avoir à lire le listing. Un exemple de plan standard pour votre rapport vous est fourni en annexe de ce document. Vous êtes libre de changer ce plan, d'y ajouter des rubriques ou même d'en retirer si vous n'avez rien de significatif à y faire apparaître.

Votre programme sera soumis à une batterie de tests semi-automatiques qui permettront de juger sa correction par rapport à l'objectif du projet. À cette occasion, vous devrez être capable de proposer des tests destinés à convaincre du bon fonctionnement de votre application, de répondre aux questions de l'enseignant de façon claire et synthétique, mais également d'éditer rapidement votre programme pour y intégrer des (petites) modifications à la demande de l'enseignant.

5.3 Dates à retenir

Remise du rapport	:	**vendredi 13 décembre 2013 à 18h**
Tests	:	**vendredi 13 décembre 2013 après-midi**

Un plan standard pour le rapport

1 Introduction

1.1 But du projet

1.2 Cahier des charges

(hypothèses de travail, tâches à réaliser)

2 Choix de conception

2.1 Structures de données et types

2.2 Algorithmes et fonctions

3 Tests

3.1 Tests unitaires

(tests de chaque fonction séparément)

3.2 Tests d'intégration

(tests de l'application principale)

3.3 Dimensionnement

4 Conclusion

4.1 Difficultés rencontrées

4.2 Bilan technique du projet

(critiques, améliorations, extensions de votre solution)

4.3 Bilan personnel du projet