Cycle préparatoire 2ème année

Projet Mastermind™

S. Arib, A. Diattara, R. Dujol, B. George, A. Lamani, D. Zouache

Matière : **Programmation fonctionnelle**	*Écheance* : **Vendredi 18 janvier 2019, 23:59**
	Nombre de pages : **5**

L'objectif de ce projet est d'implémenter une version informatique du célèbre jeu Mastermind™.

Principe du jeu

Ce jeu oppose deux joueurs. Voici le déroulement d'une partie.

1. Un des joueurs choisit un code de quatre pions à l'aide de six couleurs (il est possible d'utiliser plusieurs fois la même couleur), sans le montrer à son adversaire.
2. L'autre joueur a dix tentatives pour deviner le code.
   À chaque tentative, le créateur du code donne le nombre de pions bien placés ainsi que le nombre de pions mal placés dans la proposition par rapport au code qu'il a créé.
3. Si le code n'a pas été trouvé à l'issue des dix tentatives, le créateur du code remporte la partie. Sinon, l'autre joueur remporte la partie.

Pour des raisons d'équité, chaque joueur doit avoir le même nombre d'occasions de créer le code : on joue donc un nombre pair de parties, en alternant à tour de rôle.

Ce jeu a connu de nombreuses évolutions sur les points suivants :
— le nombre de pions par code ;
— le nombre de couleurs disponibles ;
— le remplacement de couleurs par d'autres symboles (lettres, etc…) ;
— …

1 Consignes générales

Non-respect des consignes *Tout rendu ne respectant pas scrupuleusement les consignes sera sanctionné à hauteur du non-respect desdites consignes.*

Constitution des groupes Tous les groupes de projet (trois membres maximum) doivent avoir été constitués **à l'issue de la première séance de projet**.

Contenu du rendu AREL Archive contenant le code CAML et un rapport
N'oubliez pas de vérifier votre archive!

Utilisation de git obligatoire

- L'évaluation de votre utilisation de git sera realisée par vos responsables du cours sur git respectifs comptant pour la note de TIPE (mais pas pour la note de projet).
- **À l'issue de la première séance de projet**, le responsable du cours sur git de votre campus [1] devra avoir été ajouté comme membre de votre projet avec le statut `Guest` (`Invité`).

Heure limite de rendu **Vendredi 18 janvier 2019 à 23h59** sur AREL

1. Antoine GIRARD (compte `agd`) pour Cergy et Elisabeth RANISAVLJEVIĆ (compte `erc`) pour Pau

2 Rapport

Vous devrez rendre un rapport qui présentera les éléments suivants (pas nécessairement dans cet ordre) :
— la structuration du code ;
— une présentation motivée des choix algorithmiques que vous avez fait ;
— la répartition du travail parmi les membres du groupe ;
— un bilan du projet.

3 Déroulement d'une partie

IMPORTANT. Seul le mode « PvE [2] » qui oppose un joueur humain à un joueur non-humain est obligatoire. *C'est le seul mode qui sera présenté dans ce sujet.*

Le mode « PvP [3] » qui oppose deux joueurs humains est facultatif et laissé à votre appréciation.

3.1 Le joueur humain doit deviner le code

Le code doit être généré aléatoirement [4] parmi tous les codes possibles.

Le joueur humain saisit une proposition et doit alors avoir les retours suivants :
— la réponse (nombre de pions bien placés et nombre de pions mal placés) ;
— le rappel du plateau de jeu contenant toutes les propositions faites par le joueur humain ainsi que les réponses associées (non obligatoire si il s'agissait de la dernière tentative).

3.2 Le joueur humain doit créer le code

Le joueur humain saisit le code à deviner.

Le joueur non-humain fait une proposition et vous devez implémenter les deux possibilités suivantes :
— soit la machine calcule elle-même la réponse, de sorte que toutes les tentatives s'enchaînent sans intervention de l'utilisateur ;
— soit le joueur humain saisit lui-même la réponse : à l'issue de la partie, vous devez pouvoir détecter si le joueur humain a fait une erreur (involontaire ou non) ; auquel cas, le joueur humain perd la partie.

Le joueur non-humain doit alors faire une sélection parmi les codes pour réduire les possibilités. Vous devrez implémenter au moins les deux algorithmes de sélection suivants :
— un algorithme dit « naïf » qui consiste à, après avoir enlevé tous les codes pour lesquels la réponse aurait été différente, choisir le premier de la nouvelle sélection ;
— un algorithme proposé par Donald E. KNUTH [5] en 1976 [1] qui trouve le code en cinq coups au plus.
Vous pouvez bien évidemment proposer d'autres algorithmes.

3.3 Qui commence ?

Lors du démarrage du jeu, le joueur qui crée le code est déterminé aléatoirement [4], puis alterne à chaque partie.

2. *Player versus Environment*
3. *Player versus Player*
4. Vous pourrez utiliser le module `Random` (cf. section 21.28 de la documentation officielle en pages 441 à 443).
5. Inventeur de TEX en 1978 et auteur de la série d'ouvrages de référence *The Art of Computer Programming*

4 Interface utilisateur

Votre code doit obligatoirement intégrer une interface utilisateur permettant de jouer.
La forme de cette interface n'est pas imposée : il peut s'agir une interface en mode texte ou graphique.

L'appel de l'interface se fera depuis la console CAML par la « fonction » `mastermind` :

```
mastermind : string -> int -> int -> bool -> unit
```

- le premier paramètre est le nom du joueur humain ;
- le deuxième paramètre est le nombre maximal de tentatives par partie ;
- le troisième paramètre est le nombre de parties à jouer : si ce nombre est impair, le nombre réel de parties sera augmenté de un ;
- le quatrième paramètre précise si les réponses seront calculées de manière automatique (`true`) ou fournies par le joueur humain (`false`) dans le cas où c'est au joueur humain de créer le code.

5 « Figures imposées »

5.1 Définition d'un code

La définition d'un code du jeu sera isolée dans un module `Code` **dont la signature est imposée**. Cette dernière est détaillée dans l'annexe A page suivante.

Ce module doit intégrer la possibilité de paramétrer les éléments suivants :
— nombre de pions par code ;
— nombre de couleurs possibles ;
— remplacement des couleurs par d'autres symboles (nombres, ...).

Seules les fonctions définies dans la section `sig ... end` du module `Code` sont permises pour traiter des codes lors de l'implémentation :
— des algorithmes de sélection de codes (« naïf », KNUTH, ...) ;
— de l'interface utilisateur.

5.2 Définition d'un algorithme de sélection

La définition d'un algorithme de sélection sera isolée dans un module `IA` **dont la signature est imposée**. Cette dernière est détaillée dans l'annexe B page 5.

5.3 Divers

- **Le code doit être fonctionnel sur les ordinateurs des salles machines de l'EISTI.**
- **Les commentaires doivent suivre le format `ocamldoc`** présenté en début de projet.
- Vous devez **minimiser l'utilisation de la récursivité** et **maximiser l'utilisation des fonctions du module `List`** [6], en particulier celles vues en cours.
- **Vous ne devez pas lever d'exceptions « manuellement »** et devez circonscrire au plus tôt celles qui sont créées par les fonctions et procédures fournies par CAML.

6. cf. section 21.18 de la documentation officielle en pages 412 à 416

A Signature du module `Code`

```
(** Module de definition d'un code dans le jeu Mastermind *)
module Code :
  sig
    (** Le type d'un pion *)
    type pion = (* A COMPLETER *)

    (** Le type d'un code *)
    type t = pion list

    (** Nombre de pions par code *)
    val nombre_pions : int

    (** Liste des couleurs possibles *)
    val couleurs_possibles : pion list

    (** Compare deux codes
      * @param code1 premier code a comparer
      * @param code2 second  code a comparer
      * @return 0 si les deux codes sont identiques,
                un entier positif si [code1] est strictement plus grand que [code2]
                un entier negatif si [code1] est strictement plus petit que [code2]
      *)
    val compare : t -> t -> int

    (** Conversion code vers chaine de caracteres (pour affichage)
      * @param code code a convertir
      * @return la representation en chaine de caracteres de [code]
      *)
    val string_of_code : t -> string

    (** Conversion chaine de caracteres vers code (pour saisie)
      * @param string chaine de caractere saisie
      * @return le code correspondant a la saisie si la conversion est possible
                [None] si la conversion n'est pas possible
      *)
    val code_of_string : string -> t option

    (** La liste de tous les codes permis *)
    val tous : t list

    (** La liste de toutes les reponses possibles *)
    val toutes_reponses : (int * int) list ;;

    (** Calcule la reponse d'un code par rapport au code cache
      * @param      code le code propose
      * @param vrai_code le code cache
      * @return un couple (nombre de pions bien places, nombre de pions mal places)
                [None] si la reponse ne peut etre calculee
      *)
    val reponse : t -> t -> (int * int) option
  end ;;
```

B Signature module `IA`

```
(** Algorithmes de recherche de code *)
module IA :
  sig
    (** Nombre d'algorithmes developpes *)
    val nombre_methodes : int

    (** Choisit un code a proposer
      * @param methode 0 pour l'algorithme naif,
      *                1 pour l'algorithme de KNUTH
      *                ... et ainsi de suite
      * @param essais la liste des codes deja proposes
      * @param possibles la liste des codes possibles
      * @return le prochain code a essayer
      *)
    val choix  : int -> Code.t list -> Code.t list -> Code.t

    (** Filtre les codes possibles
      * @param methode 0 pour l'algorithme naif,
      *                1 pour l'algorithme de KNUTH
      *                ... et ainsi de suite
      * @param (code, rep) le code essaye et la reponse correspondante
      * @param possibles la liste de courante de codes possibles
      * @return la nouvelle liste de codes possibles
      *)
    val filtre : int -> (Code.t * (int * int) option) -> Code.t list -> Code.t list
  end ;;
```

C Interface graphique : module `Graphics`

Il est possible de définir une interface graphique en CAML. Pour cela, il faut d'abord charger le module `Graphics` avec la directive suivante :

```
# #load "graphics.cma" ;;
```

Le module `Graphics` est alors utilisable comme tout autre module : sa documentation complète se trouve dans la documentation officielle au chapitre 26, pages 535 à 543.

Références

[1] Donald E. KNUTH : The computer as a master mind. *Journal of Recreational Mathematics*, 9(1):1 – 6, 1976-1977.

[2] Wikipedia contributors : Mastermind (board game) — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Mastermind_(board_game)`.