Rapport de projet : Mastermind en Ocaml

PRESENTÉ PAR :

MAMADOU BA

MOUHAMAD GAYE

PR : MR GAYE

UNIVERSITÉ IBA DER THIAM DE THIÈS

UFR SCIENCES ET TECHNOLOGUIES

DÉPARTEMENT INFORMATIQUE

LICENCE 3 INFORMATIQUE OPTION GÉNIE LOGICIEL

MODULE : PROGRAMMATION FONCTIONNELLE

Introduction :

Ce rapport présente le travail réalisé dans le cadre du projet de programmation fonctionnelle en OCaml, réalisé par Mamadou Ba et Mouhamad Gaye. Le but du projet est d'implémenter le jeu du Mastermind, qui est un jeu de logique où le joueur doit deviner un code secret composé de quatre couleurs parmi huit possibles. Le joueur dispose de dix tentatives (on a fixé le nombre de tentatives à 10) pour trouver le code, et à chaque tentative, il reçoit un indice sur le nombre de couleurs bien placées et mal placées.

Le rapport est organisé comme suit :

- La section 1 décrit la définition des types couleur et code, ainsi que la fonction pour générer un code aléatoire.
- La section 2 décrit la fonction pour comparer les codes et renvoyer le nombre de couleurs bien placées et mal placées.
- La section 3 décrit la fonction principale pour jouer le jeu, qui utilise les fonctions précédentes et interagit avec le joueur.
- La section 4 conclut le rapport et propose des perspectives d'amélioration.

Définition des types et génération du code secret

La première étape du projet consiste à définir les types couleur et code, qui représentent respectivement une couleur parmi les huit possibles, et une liste de quatre couleurs. On utilise pour cela un type énuméré pour les couleurs, et un type composé pour les codes. Voici le code OCaml correspondant :

```
(* Définition des types  couleur et code *)
type couleur = Cyan |  Bleu | Blanc | Vert | Magenta | Jaune | Noir | Rouge
type code = couleur list
```

La deuxième étape consiste à écrire une fonction pour générer un code aléatoire, qui sera le code secret à deviner par le joueur. On utilise pour cela la fonction Random.int du module Random, qui permet de tirer un entier aléatoire entre 0 et un nombre donné. On utilise

également la fonction List.nth du module List, qui permet d'accéder à un élément d'une liste à une position donnée. On définit une fonction récursive aux, qui prend en paramètre le nombre de couleurs à générer, et la liste des couleurs déjà générées. Si le nombre de couleurs est nul, on renvoie la liste obtenue. Sinon, on tire une couleur aléatoire parmi les huit possibles, en utilisant la liste des couleurs comme référence, et on l'ajoute au début de la liste. On rappelle ensuite la fonction aux avec le nombre de couleurs diminué de un, et la nouvelle liste. Voici le code OCaml correspondant :

```
(* Fonction pour générer un code aléatoire  cad une combinaison *)
let generer_code n couleurs =
  let rec aux n liste_combinaisonAgenerer =
    if n = 0 then liste_combinaisonAgenerer
    else
      (* cette expression couleur_aleatoire prend en argument la liste des couleur et la position de chaque couleur dans la liste *)
      let couleur_aleatoire = List.nth couleurs (Random.int (List.length couleurs)) in
      aux (n-1) (couleur_aleatoire :: liste_combinaisonAgenerer) (* cette expression permet d'ajouter  le couleur  prise dans la liste appelée
                                                                    'liste_combinaisonAgenerer' au debut de la liste   *)
  in aux n []
```

Comparaison des codes et renvoi des indices

La troisième étape du projet consiste à écrire une fonction pour comparer les codes et renvoyer le nombre de couleurs bien placées et mal placées. On utilise pour cela un filtrage par motifs sur les deux listes de couleurs, en les parcourant simultanément. On utilise également la fonction List.mem du module List, qui permet de tester si un élément appartient à une liste. On définit une fonction récursive comparerCombinaison, qui prend en paramètre les deux codes à comparer, et les nombres de couleurs bien placées et mal placées déjà trouvés. Si l'un des codes est vide, on renvoie les nombres obtenus. Sinon, on compare les premiers éléments des deux codes. S'ils sont égaux, on incrémente le nombre de couleurs bien placées, et on rappelle la fonction comparerCombinaison avec les queues des deux combinaisons. Sinon, on teste si le premier élément du code proposé appartient au code secret. Si c'est le cas, on incrémente le nombre de couleurs mal placées, et on rappelle fonction comparerCombinaison avec les queues des deux codes. Sinon, on ne change pas les nombres, et on rappelle la fonction comparerCombinaison avec les queues des deux codes. Voici le code OCaml correspondant :

```
(* Fonction pour comparer  les deux codes le code entré par le joueur et le code défini par la machine et
la fonction va retourner le nombre de couleurs bien placées et mal placées*)
let comparer_codes code_secret code_propose =
  let rec comparerCombinaison code_s code_p bien_place mal_place =
    match code_s, code_p with
    | [], _ | _, [] -> (bien_place, mal_place)
    | h1::t1, h2::t2 ->
        if h1 = h2 then comparerCombinaison t1 t2 (bien_place + 1) mal_place
        else if List.mem h2 code_s then comparerCombinaison t1 t2 bien_place (mal_place + 1)
        else comparerCombinaison t1 t2 bien_place mal_place
  in comparerCombinaison code_secret code_propose 0 0
```

Fonction principale pour jouer le jeu

La quatrième étape du projet consiste à écrire la fonction principale pour jouer le jeu, qui utilise les fonctions précédentes et interagit avec le joueur. On utilise pour cela la fonction print_endline du module Pervasives, qui permet d'afficher une chaîne de caractères suivie d'un retour à la ligne sur la sortie standard. On utilise également la fonction read_line du module Pervasives, qui permet de lire une ligne de texte depuis l'entrée standard. On utilise aussi la fonction Printf.printf du module Printf, qui permet de formater et d'afficher une chaîne de caractères selon un format donné. On utilise enfin la fonction exit du module Pervasives, qui permet de terminer le programme avec un code de sortie donné. On définit une fonction récursive jouer_tour, qui prend en paramètre le numéro de la tentative courante. Si la tentative dépasse la limite de dix, on affiche un message de défaite et on termine le programme. Sinon, on affiche le numéro de la tentative et on demande au joueur de proposer un code. On lit le code proposé par le joueur, en le découpant en mots, et en transformant chaque mot en une couleur, en utilisant un filtrage par motifs. Si le mot n'est pas une couleur reconnue, on déclenche une exception. On compare ensuite le code proposé avec le code secret, et on obtient le nombre de couleurs bien placées et mal placées. Si le nombre de couleurs bien placées est égal à la longueur du code secret, on affiche un message de victoire et on termine le programme. Sinon, on affiche le nombre de couleurs bien placées et mal placées, et on incrémente la tentative pour le prochain tour. Voici le code OCaml correspondant :

```
(* Fonction principale pour jouer le jeu *)
let jouer_mastermind () =
  let couleurs = [Rouge; Vert; Jaune; Bleu; Magenta; Blanc; Noir; Cyan] in
  let code_secret = generer_code 4 couleurs in
  let limite_tentatives = 10 in (*  le nombre maximal de tentatives possibles *)
  let rec jouer_tour tentative =
    if tentative > limite_tentatives then
      begin
        print_endline " le nombre de tentatives depassé . Vous avez perdu !";
        exit 0 (* Terminer le programme *)
      end;
    Printf.printf "Tentative %d/%d. Proposez un code : " tentative limite_tentatives;
    let code_propose = read_line () |> String.split_on_char ' ' |> List.map (fun s -> match s with
        | "Rouge" -> Rouge
        | "Vert" -> Vert
        | "Jaune" -> Jaune
        | "Bleu" -> Bleu
        | "Magenta" -> Magenta
        | "Blanc" -> Blanc
        | "Noir" -> Noir
        | "Cyan" -> Cyan
        | _ -> failwith "Couleur non reconnue") in
    let (bien_place, mal_place) = comparer_codes code_secret code_propose in
    if bien_place = List.length code_secret then begin
      print_endline "Bravo, vous avez trouvé le code !";
      exit 0 (* Terminer le programme en cas de victoire *)
    end else begin
      Printf.printf "Bien placés : %d, Mal placés : %d\n" bien_place mal_place;

       (* Incrémenter la tentative pour le prochain tour *)
      jouer_tour (tentative + 1)
    end
  in jouer_tour 1 (* Commencer avec la tentative num 1 *)
```

Pour lancer le jeu maintenant on appelle la fonction principale : voici le code correspondant :

Conclusion

Ce projet est une experience enrichissante qui nous a permis de mieux voir le concept de la programmation fonctionnelle avec le langage Ocaml . Un langage puissant avec un syntaxe peu intruitif pour ceux qui sont habitués avec la programmation procédurale et orienté objet.