Table des matières

Dans cette partie nous allons détailler les étapes d'implémentation de l'application.

4.1 Compilation

Afin de générer les fichiers binaires, nous avons créé un Makefile très générique paramétrable à souhait. Il semblerait que les performances d'exécution soit optimisées en utilisant le compilateur ocamlopt. Ce qui rend le choix intéressant dans un cas d'utilisation comme le nôtre ou le programme est sujet à des explosions combinatoires.

Pour une compilation standard avec `ocamlc`, utiliser : `make`

```
# Pour recompiler le système progressivement :
#     make
# Pour recalculer les dépendances entre les modules :
#     make depend
# Pour supprimer l'exécutable et les fichiers compilés :
#     make clean
# Pour compiler avec le compileur de code natif
#     make opt
```

FIGURE 1: Utilisation du Makefile

4.2 Utilisation

Afin de lancer l'application, tapez : `./othello`

Il est possible de spécifier des arguments à l'exécutable qui vont influencer le déroulement du jeu. Vous pouvez obtenir la liste ces options avec :

```
$ ./othello --help
othello
  -size <int> : Taille d'une case en pixels
  -ia <bool> : Intelligence artificielle on/off
               | true  -> Minimax ab
               | false -> Case aléatoire
  -depth <int> : Profondeur maximale d'exploration
                 de l'arbre des possibilités Minimax
  -background <int> <int> <int> : Couleur du fond (RGB)
  -help : Afficher cette liste d'options
  --help : Afficher cette liste d'options
```

FIGURE 2: Utilisation de l'exécutable

Les options par défaut de l'application sont :

```
1 val cell_size : int ref = {contents = 50}
2 val bg_r : int ref = {contents = 50}
3 val bg_g : int ref = {contents = 150}
4 val bg_b : int ref = {contents = 50}
5 val size : int ref = {contents = 8}
6 val ia : bool ref = {contents = true}
7 val depth : int ref = {contents = 4}
```

FIGURE 3: Configuration par défaut

4.3 Liste exhaustive des prototypes

4.4 Algorithmes intéressants

4.4.1 Direction Légale

Methode de test de direction légal => Vrai si la direction est légale

```
(* Methode de test de direction légal => Vrai si la direction est légale *)
let playable_dir board c (x, y) (dx, dy) =
  let rec playable_dir_rec (x, y) valid =
    if not (check_pos board x y) then
      false
    else (
      match board.(x).(y) with
        | Empty -> false
        | cell ->
          if cell = (get_opponent c) then
            playable_dir_rec (x + dx, y + dy) true
          else
            valid
    )
    in playable_dir_rec (x + dx, y + dy) false
;;
```

FIGURE 4: Direction légale

4.4.2 Coup Légal

```
(* Methode de test de coup légal => Vrai si le coup est légal *)
let playable_cell board c x y =
  if not (check_pos board x y) then
    false
  else (
    let directions = [
      (-1, -1); (-1, 0); (-1, 1);
      (0 , -1); (* X *)  (0 , 1);
      (1 , -1); (1 , 0); (1 , 1)
    ]
    in match board.(x).(y) with
      | Empty -> ( true && (
     List.fold_left
       (fun a b -> a || b)
       false
              (List.map
                (fun d -> playable_dir board c (x, y) d)
                directions
              )
            )
         )
      | _ -> false
  )
;;
```

FIGURE 5: Coup légal

4.4.3 Simulation de jeu

```
(* Méthode pour simuler le jeu sur une case *)
let sim_play_cell board c x y =
  let sim_board = (copy_board board) in
  let directions = [
    (-1, -1); (-1, 0); (-1, 1);
    (0 , -1); (* X *)  (0 , 1);
    (1 , -1); (1 , 0); (1 , 1)
  ]
  and opponent = (get_opponent c)
  in (
    List.iter
      (fun (dx, dy) ->
        if (playable_dir sim_board c (x, y) (dx, dy)) then
          let rec take (x, y) =
            if (check_pos sim_board x y) then
            if (sim_board.(x).(y) = opponent) then (
              sim_board.(x).(y) <- c;
              take (x + dx, y + dy)
            )
          in take (x + dx, y + dy)
      )
      directions
  );
  sim_board.(x).(y) <- c;
  sim_board
;;
```

FIGURE 6: Simuler un coup