

Ce que nous avons vu...

Programmation Fonctionnelle

On emballe

David Teller

05/02/2007

- ▶ Lancement et utilisation d'OCaml
- ▶ fonctions, valeurs
- ▶ listes, tuples, enregistrements et autre types et structures de données
- ▶ modules.

Ce qui nous manque

- ▶ des exemples
- ▶ de la documentation
- ▶ des exceptions
- ▶ des labels
- ▶ des traits impératifs
- ▶ des bibliothèques

Au programme du jour

- exceptions** quand, pourquoi et comment arrêter un algorithme
- labels** comment ne plus jamais se tromper dans l'ordre des arguments d'une fonction
- un exemple** dont vous êtes le héros

Au programme du jour

- exceptions** quand, pourquoi et comment arrêter un algorithme
- labels** comment ne plus jamais se tromper dans l'ordre des arguments d'une fonction
- un exemple** dont vous êtes le héros
 - ... et probablement des bouts de bibliothèques

Interlude

Exceptions
Bases
Assertions

Labels

Un exemple dont vous êtes les héros

Exemple à construire

Conclusions

Exercice 8

Q Écrire une fonction qui ajoute des éléments à la fin d'une liste.

Exemples

```
* ajoute_a_la_fin [1;2;3;4;5] 6 ::
- : int list = [1;2;3;4;5;6]
* ajoute_a_la_fin [] "Bleu" ::
- : string list = ["Bleu"]
```

Solution possible

On s'attend à ce que le type soit 'a list -> 'a -> 'a list

Solution possible

On s'attend à ce que le type soit `'a list -> 'a -> 'a list`

```
let rec ajoute_a_la_fin liste e = match liste with
| [] -> [e]
| h::t -> h::(ajoute_a_la_fin t e)::
```



Exercice 9

Q Écrire une fonction qui extraira d'une liste tous les éléments contenant une propriété donnée.

Exemples

```
* extrait [1;2;0;-5;7;0] (( > ) 0)::
- : int list = [1;2]
* extrait [1;2;0;-5;7;0] (( < ) 0)::
- : int list = [-5;7;0]
* extrait [1;2;0;-5;7;0] (( = ) 0)::
- : int list = []
```



Solutions possibles

Solution simple

```
let rec extract liste f = match liste with
| [] -> []
| h::t when f h -> h::(extract t f)
| h::t -> extract t f
```



Solutions possibles

Solution simple

```
let rec extract liste f = match liste with
| [] -> []
| h::t when f h -> h::(extract t f)
| h::t -> extract t f
```

Solution capillotracée

```
let extract liste f =
  List.fold_left (fun l e -> if f e then e::l else l) [] liste::
```



Prochains exercices

...et encore un !

Exercice à rendre Écrire une fonction qui insère un élément à sa place dans une liste ordonnée, c'est-à-dire de manière à obtenir une nouvelle liste ordonnée.

Exemple

```
* insere 'a' ['a';'b';'c';'d';'e';'f';'g';'h';'i';'j';'k';'l';'m';'n';'o';'p';'q';'r';'s';'t';'u';'v';'w';'x';'y';'z'];;
- : char list = ['a';'b';'c';'d';'e';'f';'g';'h';'i';'j';'k';'l';'m';'n';'o';'p';'q';'r';'s';'t';'u';'v';'w';'x';'y';'z'];;
* insere 7 [1;2;3;4;5;6;7;8];;
- : int list = [1;2;3;4;5;6;7;8]
```

Exercice à rendre À partir de la fonction précédente, écrire une fonction qui trie une liste.

Exemple

```
* trie_liste [6;7;8;3;4;2];;
- : int list = [2;3;4;6;7;8]
```



Interlude

Exceptions

Bases

Assertions

Labels

Un exemple dont vous êtes les héros

Exemple à construire

Conclusions

Exceptionnellement

OCaml dispose d'un système d'exceptions proche de celui de Java, juste un peu plus simple.



Exceptionnellement

OCaml dispose d'un système d'exceptions proche de celui de Java, juste un peu plus simple.
En fait, le système d'exceptions de Java est une simplification de celui de C++, qui est une version plus compliquée de celui de ML.



Kezako ?

Une exception est un signal que vous (ou une bibliothèque de programmation) envoyez au reste du programme dans des cas exceptionnels :

- ▶ un problème externe rend inutile, impossible ou nuisible de continuer l'algorithme en cours (problèmes de disque, mauvais mot de passe, l'utilisateur a mal formulé sa question...)
- ▶ un succès rend inutile de continuer l'algorithme en cours (vous avez trouvé la solution sans avoir à faire tous les calculs)
- ▶ une erreur de logique signifie que le programme est incorrect (votre programme a atteint un état qui devrait être impossible).



Exceptionnellement

OCaml dispose d'un système d'exceptions proche de celui de Java, juste un peu plus simple.
En fait, le système d'exceptions de Java est une simplification de celui de C++, qui est une version plus compliquée de celui de ML.

Exemple

```
* 1 / 0 ::  
Uncaught exception : Division_by_zero
```



Kezako ?

Une exception est un signal que vous (ou une bibliothèque de programmation) envoyez au reste du programme dans des cas exceptionnels :

- ▶ un problème externe rend inutile, impossible ou nuisible de continuer l'algorithme en cours (problèmes de disque, mauvais mot de passe, l'utilisateur a mal formulé sa question...)
- ▶ un succès rend inutile de continuer l'algorithme en cours (vous avez trouvé la solution sans avoir à faire tous les calculs)
- ▶ une erreur de logique signifie que le programme est incorrect (votre programme a atteint un état qui devrait être impossible).

Dans tous ces cas, il est nécessaire d'arrêter l'algorithme – et peut-être le programme lui-même.



Déclenchons une exception

On lance une exception à l'aide de la fonction raise.
 L'exception Failure sert à représenter les erreurs fatales.

```
# let division p q =
  if q = 0 then
    raise (Failure "Division par zéro!")
  else
    p / q ;;
# division 5 2;;
- : int = 2
# division 5 0;;
Exception: Failure "Division par zéro!".
```

Alors, qu'est-ce que c'est qu'une exception ?

```
# raise ;;
- : exn -> 'a = <fst>
# 1 / 0 ;;
Uncaught exception: Division_by_zero
# Division_by_zero ;;
- : exn = Division_by_zero
```

Alors, qu'est-ce que c'est qu'une exception ?

```
# raise ;;
- : exn -> 'a = <fst>
# 1 / 0 ;;
Uncaught exception: Division_by_zero
# Division_by_zero ;;
- : exn = Division_by_zero
```

Manifestement, c'est un constructeur du type exn. Et raise, c'est une fonction.

Alors, qu'est-ce que c'est qu'une exception ?

```
# raise ;;
- : exn -> 'a = <fst>
# 1 / 0 ;;
Uncaught exception: Division_by_zero
# Division_by_zero ;;
- : exn = Division_by_zero
```

Manifestement, c'est un constructeur du type exn. Et raise, c'est une fonction.

Q Pourquoi le type de raise est-il exn -> 'a ?

Alors, qu'est-ce que c'est qu'une exception ?

```
# raise ::
- : exn -> 'a = <fex>
# ! / 0 ::
Uncaught exception: Division_by_zero
# Division_by_zero ::
- : exn = Division_by_zero
```

Manifestement, c'est un constructeur du type `exn`. Et `raise`, c'est une fonction.

Q Pourquoi le type de `raise` est-il `exn -> 'a` ?

A Parce que `raise` arrête l'exécution de l'algorithme – et ne renvoie donc aucune valeur.

Autres types d'exceptions

Vous pouvez définir vous-même de nouveaux types d'exceptions, à l'aide du mot-clef `exception`.

```
# exception MonException of string * int ::
exception MonException of string * int
```

Autres types d'exceptions

Vous pouvez définir vous-même de nouveaux types d'exceptions, à l'aide du mot-clef `exception`.

```
# exception MonException of string * int ::
exception MonException of string * int
```

La syntaxe est la même que pour les constructeurs de types.

Autres types d'exceptions

Vous pouvez définir vous-même de nouveaux types d'exceptions, à l'aide du mot-clef `exception`.

```
# exception MonException of string * int ::
exception MonException of string * int
```

La syntaxe est la même que pour les constructeurs de types. Typiquement, deux erreurs non-fatales différentes auront droit à des exceptions différentes, histoire de pouvoir les gérer séparément.

Gestion des exceptions

```
# let ma_division p q =
  try
    division p q
  with
    | Division_by_zero -> 0
    | Failure T -> print_endline f; raise (Failure f)::
- : int -> int -> int = <fex>
# ma_division 0 0;;
Division par zero !
Exception : Failure "Division par zero!".
```

```
# let ma_division p q =
  try
    division p q
  with
    | Division_by_zero -> 0
    | Failure T -> print_endline f; raise (Failure f)::
- : int -> int -> int = <fex>
# ma_division 0 0;;
Division par zero !
Exception : Failure "Division par zero!".
```

OCaml essaye d'exécuter tout ce qui est entre `try` et `with`. En cas d'*exception*, ce bloc est immédiatement arrêté et OCaml cherche un gestionnaire d'exception. Si celui-ci est trouvé, il est exécuté. Dans le cas contraire, l'exception est re-soulevée. Pour choisir entre les gestionnaires d'exception, une fois de plus, OCaml utilise le filtrage par motifs.

Rappelons-le

Les exceptions peuvent aussi être utilisées pour interrompre un un algorithme en cas de succès.

```
# exception Divisible of int;;
exception Divisible of int
# let cherche_diviseur_de p =
  try
    for i = 2 to int_of_float (sqrt (float_of_int p)) do
      if p mod i = 0 then
        raise (Divisible i)
    done;
    None
  with
    Divisible i -> Some i;;
val cherche_diviseur_de : int -> int option = <fex>
```

Assertions

L'une des utilisations les plus courantes des exceptions est sous la forme d'*assertions* : vérifier qu'une condition est remplie et arrêter immédiatement le programme si ce n'est pas le cas.

Assertions

L'une des utilisations les plus courantes des exceptions est sous la forme d'*assertions* : vérifier qu'une condition est remplie et arrêter immédiatement le programme si ce n'est pas le cas.

L'appel de fonction `assert e` évalue `e`. Si `e` est vrai, on continue. Sinon, le programme s'arrête avec un message d'erreur qui précise à quel endroit du programme on était.



Assertions

L'une des utilisations les plus courantes des exceptions est sous la forme d'*assertions* : vérifier qu'une condition est remplie et arrêter immédiatement le programme si ce n'est pas le cas.

L'appel de fonction `assert e` évalue `e`. Si `e` est vrai, on continue. Sinon, le programme s'arrête avec un message d'erreur qui précise à quel endroit du programme on était.

Q Mais à quoi ça sert ?

A On utilise les assertions durant la phase de développement du logiciel, pour vérifier que les pré- et post-conditions des fonctions sont correctes.



Assertions

L'une des utilisations les plus courantes des exceptions est sous la forme d'*assertions* : vérifier qu'une condition est remplie et arrêter immédiatement le programme si ce n'est pas le cas.

L'appel de fonction `assert e` évalue `e`. Si `e` est vrai, on continue. Sinon, le programme s'arrête avec un message d'erreur qui précise à quel endroit du programme on était.

Q Mais à quoi ça sert ?



Assertions

L'une des utilisations les plus courantes des exceptions est sous la forme d'*assertions* : vérifier qu'une condition est remplie et arrêter immédiatement le programme si ce n'est pas le cas.

L'appel de fonction `assert e` évalue `e`. Si `e` est vrai, on continue. Sinon, le programme s'arrête avec un message d'erreur qui précise à quel endroit du programme on était.

Q Mais à quoi ça sert ?

A On utilise les assertions durant la phase de développement du logiciel, pour vérifier que les pré- et post-conditions des fonctions sont correctes. C'est ce qu'on appelle la *programmation défensive* : vérifier ses hypothèses durant l'exécution du programme, à coups de `assert`.



Assertions

L'une des utilisations les plus courantes des exceptions est sous la forme d'*assertions* : vérifier qu'une condition est remplie et arrêter immédiatement le programme si ce n'est pas le cas.

L'appel de fonction `assert e` évalue `e`. Si `e` est vrai, on continue. Sinon, le programme s'arrête avec un message d'erreur qui précise à quel endroit du programme on était.

Q Mais à quoi ça sert ?

A On utilise les assertions durant la phase de développement du logiciel, pour vérifier que les pré- et post-conditions des fonctions sont correctes.

C'est ce qu'on appelle la *programmation défensive* : vérifier ses hypothèses durant l'exécution du programme, à coups de `assert`.

Q Des exemples ?

Cocorico

Les assertions existent aussi dans Java. Elles ne viennent pas de ML ou OCaml.

Par contre, elles viennent d'Eiffel, un autre langage français.

Cocorico

Les assertions existent aussi dans Java. Elles ne viennent pas de ML ou OCaml.

Les assertions existent aussi dans Java. Elles ne viennent pas de ML ou OCaml.

Par contre, elles viennent d'Eiffel, un autre langage français.

Comment ça, chauvin, ce cours ?

Cocorico

Interlude

Exceptions

Bases

Assertions

Labels

Un exemple dont vous êtes les héros

Exemple à construire

Conclusions

Sans labels

Considérons les exemples suivants :

```
* let prendre_dans_matrice_1 m i j =
  m.(i).(j);;
val prendre_dans_matrice_1 : 'a array array -> int -> int -> 'a = <fex>
* let prendre_dans_matrice_2 m i j =
  m.(j).(i);;
val prendre_dans_matrice_2 : 'a array array -> int -> int -> 'a = <fex>
* let diviser_1 x y =
  x /. y;;
val diviser_1 : float -> float -> float = <fex>
* let diviser_2 x y =
  y /. x;;
val diviser_2 : float -> float -> float = <fex>
```

Tant qu'on a le code source, tout va bien...

...ni code

Une fois que le code source est parti...

```
* prendre_dans_matrice_1 ;;
- : 'a array array -> int -> int -> 'a = <fex>
* prendre_dans_matrice_2 ;;
- : 'a array array -> int -> int -> 'a = <fex>
* diviser_1 ;;
- : float -> float -> float = <fex>
* diviser_2 ;;
- : float -> float -> float = <fex>
```

...ni code

Une fois que le code source est parti...

```
* prendre_dans_matrice_1 ;;
- : 'a array array -> int -> int -> 'a = <fex>
* prendre_dans_matrice_2 ;;
- : 'a array array -> int -> int -> 'a = <fex>
* diviser_1 ;;
- : float -> float -> float = <fex>
* diviser_2 ;;
- : float -> float -> float = <fex>
```

Alors, quel argument correspond aux lignes ? Quel argument correspond aux colonnes ? Quel est le numérateur ? Quel est le dénominateur ?

Labels à la rescousse

```
# let prendre_dans_matrice_1 m ~ligne:i ~colonne:j =
  m.(i).(j);;
val prendre_dans_matrice_1 : 'a array array -> ligne:int -> colonne:int -> 'a =
<fex>
# let prendre_dans_matrice_2 m ~colonne:i ~ligne:j =
  m.(j).(i);;
val prendre_dans_matrice_2 : 'a array array -> colonne:int -> ligne:int -> 'a =
<fex>
# let diviser_1 x ~par:y =
  x /. y;;
val diviser_1 : float -> par:float -> float = <fex>
# let diviser_2 ~par:x y =
  y /. x;;
val diviser_2 : float -> par:float -> float = <fex>
```

Labels à la rescousse

```
# let prendre_dans_matrice_1 m ~ligne:i ~colonne:j =
  m.(i).(j);;
val prendre_dans_matrice_1 : 'a array array -> ligne:int -> colonne:int -> 'a =
<fex>
# let prendre_dans_matrice_2 m ~colonne:i ~ligne:j =
  m.(j).(i);;
val prendre_dans_matrice_2 : 'a array array -> colonne:int -> ligne:int -> 'a =
<fex>
# let diviser_1 x ~par:y =
  x /. y;;
val diviser_1 : float -> par:float -> float = <fex>
# let diviser_2 ~par:x y =
  y /. x;;
val diviser_2 : float -> par:float -> float = <fex>
```

Vous avez vu la différence ?

...sans les mains !

```
# prendre_dans_matrice_1;;
- : 'a array array -> ligne:int -> colonne:int -> 'a = <fex>
# prendre_dans_matrice_2;;
- : 'a array array -> colonne:int -> ligne:int -> 'a = <fex>
# diviser_1;;
- : float -> par:float -> float = <fex>
# diviser_2;;
- : float -> par:float -> float = <fex>
```

...sans les mains !

```
# prendre_dans_matrice_1;;
- : 'a array array -> ligne:int -> colonne:int -> 'a = <fex>
# prendre_dans_matrice_2;;
- : 'a array array -> colonne:int -> ligne:int -> 'a = <fex>
# diviser_1;;
- : float -> par:float -> float = <fex>
# diviser_2;;
- : float -> par:float -> float = <fex>
```

Et pour utiliser les fonctions

```
# prendre_dans_matrice_1 ~ligne:0 ~colonne:0;;
- : 'a array array -> 'a = <fex>
# diviser_1 5.0 ~par:2.0;;
- : float = 2.5
```

À retenir

Pour que l'argument x porte un label `lab` : `lab.x`.

Pour appeler une fonction en donnant au label `lab` la valeur `5` : `lab.5`.

Les labels, c'est chouette

Avec les labels

- ▶ vous pouvez changer l'ordre des paramètres de votre fonction sans changer le résultat
- ▶ vous pouvez toujours currier et décurrier (i.e. appliquer une fonction à seulement certains de ses arguments) – et avec plus de liberté
- ▶ vous n'êtes pas obligés de préciser le nom du label, si vous conservez l'ordre originel des paramètres
- ▶ vous pouvez aussi introduire des arguments optionnels – nous en parlerons (peut-être) une autre fois.

Les labels, c'est chouette

Avec les labels

- ▶ vous pouvez changer l'ordre des paramètres de votre fonction sans changer le résultat
- ▶ vous pouvez toujours currier et décurrier (i.e. appliquer une fonction à seulement certains de ses arguments) – et avec plus de liberté
- ▶ vous n'êtes pas obligés de préciser le nom du label, si vous conservez l'ordre originel des paramètres
- ▶ vous pouvez aussi introduire des arguments optionnels – nous en parlerons (peut-être) une autre fois.

Les labels sont très utiles dès que plusieurs paramètres ont le même type mais pas le même rôle.

Les labels, c'est chouette

Avec les labels

- ▶ vous pouvez changer l'ordre des paramètres de votre fonction sans changer le résultat
- ▶ vous pouvez toujours currier et décurrier (i.e. appliquer une fonction à seulement certains de ses arguments) – et avec plus de liberté
- ▶ vous n'êtes pas obligés de préciser le nom du label, si vous conservez l'ordre originel des paramètres
- ▶ vous pouvez aussi introduire des arguments optionnels – nous en parlerons (peut-être) une autre fois.

Les labels sont très utiles dès que plusieurs paramètres ont le même type mais pas le même rôle.

Pour le coup, ça ne vient pas d'un langage français, mais de SmallTalk (Xerox).

Les labels, c'est chouette

Avec les labels

- ▶ vous pouvez changer l'ordre des paramètres de votre fonction sans changer le résultat
- ▶ vous pouvez toujours currier et décorrier (i.e. appliquer une fonction à seulement certains de ses arguments) – et avec plus de liberté
- ▶ vous n'êtes pas obligés de préciser le nom du label, si vous conservez l'ordre originel des paramètres
- ▶ vous pouvez aussi introduire des arguments optionnels – nous en parlerons (peut-être) une autre fois.

Les labels sont très utiles dès que plusieurs paramètres ont le même type mais pas le même rôle.

Pour le coup, ça ne vient pas d'un langage français, mais de SmallTalk (Xerox).

Mangez-en quand même, c'est bon.



Interlude

Exceptions
Bases
Assertions

Labels

Un exemple dont vous êtes les héros

Exemple à construire

Conclusions



1. Former un binôme. ✓
2. Choisir un sujet. ✓
3. Préciser le sujet et les objectifs.
4. Procéder à des recherches bibliographiques.
5. Découper le sujet en problèmes, sous-problèmes, etc.
6. Répartir les sous-problèmes dans le binôme.
7. Répartir les sous-problèmes en modules.
8. Définir les interfaces des modules.
9. Concevoir les protocoles de tests.
10. Concevoir les algorithmes.
11. Implanter les algorithmes.
12. Tester.
13. Corriger les bugs.
14. Documenter.
15. Ajouter des fonctionnalités et retourner en 3.
16. Rédiger le rapport.



Énoncé

Projet Résolution et affichage des résultats du problème des n reines.



Énoncé

Projet Résolution et affichage des résultats du problème des n reines.

Q Que faites-vous ?



Bibliographie

Google "Il s'agit de placer n reines sur un échiquier ($n \times n$) sans qu'elles se menacent. Deux reines ne doivent pas se trouver sur la même ligne, sur la même colonne ou sur la même diagonale."



Bibliographie

Google "Il s'agit de placer n reines sur un échiquier ($n \times n$) sans qu'elles se menacent. Deux reines ne doivent pas se trouver sur la même ligne, sur la même colonne ou sur la même diagonale."

Q Et maintenant ?



Bibliographie

Google "Il s'agit de placer n reines sur un échiquier ($n \times n$) sans qu'elles se menacent. Deux reines ne doivent pas se trouver sur la même ligne, sur la même colonne ou sur la même diagonale."

Q Et maintenant ?

Q Quels sont les objectifs exacts de votre logiciel ?



Bibliographie

Google "Il s'agit de placer n reines sur un échiquier ($n \times n$) sans qu'elles se menacent. Deux reines ne doivent pas se trouver sur la même ligne, sur la même colonne ou sur la même diagonale."

Q Et maintenant ?

Q Quels sont les objectifs exacts de votre logiciel ?

- ▶ Recherche une seule solution ? Toutes les solutions ?
- ▶ Sauvegarder une image ? Afficher une liste de positions ? Afficher un dessin moche ? Afficher une interface graphique ? En trois dimensions ?
- ▶ Le logiciel doit-il tirer ses paramètres d'un fichier ? De la ligne de commande ? Avoir une interface graphique ? Doit-il être téléguidé depuis OCaml ?



1. Former un binôme. ✓
2. Choisir un sujet. ✓
3. Préciser le sujet et les objectifs. ✓
4. Procéder à des recherches bibliographiques.
5. Découper le sujet en problèmes, sous-problèmes, etc.
6. Répartir les sous-problèmes dans le binôme.
7. Répartir les sous-problèmes en modules.
8. Définir les interfaces des modules.
9. Concevoir les protocoles de tests.
10. Concevoir les algorithmes.
11. Planter les algorithmes.
12. Tester.
13. Corriger les bugs.
14. Documenter.
15. Ajouter des fonctionnalités et retourner en 3.
16. Rédiger le rapport.



Recherches bibliographiques

Wikipedia "Un algorithme à retour sur traces essaye chaque possibilité jusqu'à ce qu'il trouve la bonne. Pendant sa recherche, si une alternative ne fonctionne pas, l'algorithme revient sur ses pas jusqu'à son dernier choix et essaye l'alternative suivante. Si toutes les alternatives sont épuisées, la recherche retourne au choix encore précédent et essaye l'alternative suivante. S'il n'y a plus de points de choix, il n'y a pas de solution."



Recherches bibliographiques

Wikipedia "Un algorithme à retour sur traces essaye chaque possibilité jusqu'à ce qu'il trouve la bonne. Pendant sa recherche, si une alternative ne fonctionne pas, l'algorithme revient sur ses pas jusqu'à son dernier choix et essaye l'alternative suivante. Si toutes les alternatives sont épuisées, la recherche retourne au choix encore précédent et essaye l'alternative suivante. S'il n'y a plus de points de choix, il n'y a pas de solution."

Google OCaml dispose des bibliothèques d'affichage Graphics (dessin minimaliste de points et de lignes), OCamlSDL (affichage haute performance pour applications multimedia), MLGame (dessins style jeux vidéos 2D), LablGTK (interfaces graphiques), LablGL (affichage 3D), Cairo (affichage vectoriel et sauvegarde d'images), OCamlImage (affichage, lecture et sauvegarde d'images) et la bibliothèque ImageMagick (lecture et manipulation d'images).



Recherches bibliographiques

Wikipedia "Un algorithme à retour sur traces essaye chaque possibilité jusqu'à ce qu'il trouve la bonne. Pendant sa recherche, si une alternative ne fonctionne pas, l'algorithme revient sur ses pas jusqu'à son dernier choix et essaye l'alternative suivante. Si toutes les alternatives sont épuisées, la recherche retourne au choix encore précédent et essaye l'alternative suivante. S'il n'y a plus de points de choix, il n'y a pas de solution."

Google OCaml dispose des bibliothèques d'affichage Graphics (dessin minimaliste de points et de lignes), OCamlSDL (affichage haute performance pour applications multimedia), MLGame (dessins style jeux vidéos 2D), LablGTK (interfaces graphiques), LablGL (affichage 3D), Cairo (affichage vectoriel et sauvegarde d'images), OCamlImage (affichage, lecture et sauvegarde d'images) et la bibliothèque ImageMagick (lecture et manipulation d'images).

Hypothèse A priori, les recherches bibliographiques ne devraient pas vous en apprendre beaucoup plus sur le problème lui-même.



1. Former un binôme. ✓
2. Choisir un sujet. ✓
3. Préciser le sujet et les objectifs. ✓
4. Procéder à des recherches bibliographiques. ✓
5. Découper le sujet en problèmes, sous-problèmes, etc.
6. Répartir les sous-problèmes dans le binôme.
7. Répartir les sous-problèmes en modules.
8. Définir les interfaces des modules.
9. Concevoir les protocoles de tests.
10. Concevoir les algorithmes.
11. Planter les algorithmes.
12. Tester.
13. Corriger les bugs.
14. Documenter.
15. Ajouter des fonctionnalités et retourner en 3.
16. Rédiger le rapport.



Et ensuite ?

Q Et maintenant ?

Et ensuite ?

Q Et maintenant ?

A Trois problèmes principaux :

- ▶ recherche d'une solution quelconque pour n reines sur un échiquier $n \times n$
- ▶ "lire" n (depuis l'interface graphique, la ligne de commande...)
- ▶ afficher une solution.



Découpage en outils

compilation un outil de compilation qui compile les modules dans le bon ordre et produit un exécutable

auto-test un outil qui va essayer automatiquement un certain nombre de n possibles et, à chaque fois, vérifier que la solution fonctionne bien

le **programme** le logiciel lui-même



1. Former un binôme. ✓
2. Choisir un sujet. ✓
3. Préciser le sujet et les objectifs. ✓
4. Procéder à des recherches bibliographiques. ✓
5. Découper le sujet en problèmes, sous-problèmes, etc. ✓
6. Répartir les sous-problèmes dans le binôme. ✓
7. Répartir les sous-problèmes en modules. ✓
8. Définir les interfaces des modules.
9. Concevoir les protocoles de tests.
10. Concevoir les algorithmes.
11. Implanter les algorithmes.
12. Tester.
13. Corriger les bugs.
14. Documenter.
15. Ajouter des fonctionnalités et retourner en 3.
16. Rédiger le rapport.



Découpage en modules

Solutions à partir d'un n quelconque, trouve une solution / à partir d'une solution, vérifie que la solution est valide (pour le protocole de tests)

Vue affiche une solution (en utilisant `Solutions.solution`) / affiche un rapport d'erreur

Controle consulte l'environnement ou/et l'interface graphique et détermine n

Principal utilise `Controle` pour déterminer n , envoie ce n à `Solutions`



Module Vue

Module `Vue`, fichier `vue.ml`.

Types `Vue.t`, le type d'un affichage.

Fonctions

- ▶ `val Vue.init : string -> Vue.t` initialise l'affichage
- ▶ `val Vue.affiche_solution : Reines.t -> unit` - effet de bord
- ▶ `val Vue.affiche_erreur : string -> unit` - effet de bord



Module Vue

Module Vue, fichier vue.ml.

Types Vue.t, le type d'un affichage.

Fonctions

- ▶ val Vue.init : string -> Vue.t initialise l'affichage
- ▶ val Vue.affiche_solution : Reines.t -> unit - effet de bord
- ▶ val Vue.affiche_erreur : string -> unit - effet de bord

Note Pour le moment, ceci est totalement indépendant de la nature de la vue : affichage 2D, 3D, fichier...

Module Controle

Module Controle, fichier controle.ml.

Fonctions

- ▶ val Controle.init : unit -> Controle.t
- ▶ val Controle.get_n : unit -> int

Module Controle

Module Controle, fichier controle.ml.

Fonctions

- ▶ val Controle.init : unit -> Controle.t
- ▶ val Controle.get_n : unit -> int

Note Pour le moment, ceci est totalement indépendant de la nature du contrôle : ligne de commande, Gtk...

Module Solutions

Module Solutions, fichier solutions.ml.

Types Solutions.t, le type d'une solution.

Fonctions

- ▶ val Solutions.to_list : Solutions.t -> (int * int) list
- ▶ val cherche : int -> Solutions.t
- ▶ val verifie : Solutions.t -> bool

Module Solutions

Module Solutions, fichier solutions.ml.

Types Solutions.t, le type d'une solution.

Fonctions

- ▶ val Solutions.to_list : Solutions.t -> (int * int) list
- ▶ val cherche : int -> Solutions.t
- ▶ val verifie : Solutions.t -> bool

Note En pratique, Solutions.t sera probablement égal à (int * int) list, mais bon, il n'est pas exclus qu'on trouve une meilleure représentation plus tard. Du coup, autant lui donner un autre nom. Par convention, Solutions.t.



Module Principal

Module Reines, fichier reines.ml.

Valeurs

- ▶ val go : unit



Module Principal

Module Reines, fichier reines.ml.

Valeurs

- ▶ val go : unit

Note On aurait pu choisir n'importe quel nom pour le programme principal. La valeur de go est forcément () car tout ce qui compte est que go soit évalué.

1. Former un binôme. ✓
2. Choisir un sujet. ✓
3. Préciser le sujet et les objectifs. ✓
4. Procéder à des recherches bibliographiques. ✓
5. Découper le sujet en problèmes, sous-problèmes, etc. ✓
6. Répartir les sous-problèmes dans le binôme. ✓
7. Répartir les sous-problèmes en modules. ✓
8. Définir les interfaces des modules. ✓
9. Concevoir les protocoles de tests.
10. Concevoir les algorithmes.
11. Planter les algorithmes.
12. Tester.
13. Corriger les bugs.
14. Documenter.
15. Ajouter des fonctionnalités et retourner en 3.
16. Rédiger le rapport.



Etc.

Si le temps le permet, j'implanterai ce projet d'ici la semaine prochaine et je vous montrerai le résultat.

Interlude

Exceptions
Bases
Assertions

Labels

Un exemple dont vous êtes les héros

Exemple à construire

Conclusions

Énoncé

Projet Génération, résolution et affichage de labyrinthes.

Énoncé

Projet Génération, résolution et affichage de labyrinthes.
Q Que faites-vous ?

Énoncé

La prochaine fois...

Projet Génération, résolution et affichage de labyrinthes.

Q Que faites-vous ?

Cette fois, c'est sans filet (et sans transparents).

- ▶ quelques traits impératifs
- ▶ de la compilation, beaucoup de compilation et encore de la compilation.