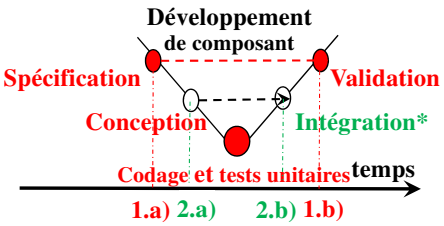
UNIVERSITÉ PARIS DESCARTES

SDA - Structures de Données et Algorithmes

Equipe pédagogique
Marie-José Caraty, Denis Poitrenaud, Julien Rossit,
Camille Kurtz, Jacques Alès-Bianchetti, Denis Jeanneau

Dédale sur un ruban de Möbius
Les cinq Sprints de Développement

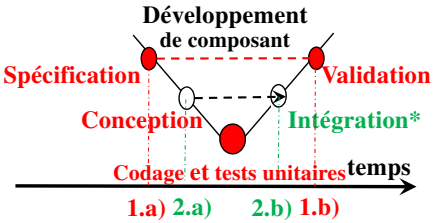
Développement agile



Choix d'un développement agile
Développement par Sprints,
chacun représentant un incrément de fonctionnalité de l'application

Cinq sprints sont définis et pour chacun des Sprints :
1) une analyse fonctionnelle
2) une spécification
3) un codage correspondant
4) un test de validation

Validation d'un Sprint



Principe de validation d'un **`Sprint#n`** à partir :
Jeu de Données de Test (JDT) : `in.txt`
et des sorties attendues : `outSPn.txt`

L'exécution de l'exécutable du **`Sprint#n`** par redirection
- des entrées à partir de `in.txt`
- des sorties vers `run.txt`

Si **`outSPn.txt`** coïncide avec **`run.txt`** :
- le **`Sprint#n`** est 0-défaut, il est validé
- vous pouvez passer au Sprint suivant : **`Sprint#n+1`**

Développement par Sprint 1/2

Attention : Chaque Sprint donne lieu à un développement matérialisé par un **projet à conserver**. **Vous serez notés sur le Sprint atteint de plus haut niveau.**

Lorsque vous passez d'un `Sprint#i` à un `sprint#i+1`, vous devez conserver intègres les sources du `Sprint#i`. Créez un **nouveau projet** où vous recopierez toutes les entités logicielles (**utiles**) du précédent `Sprint#i`, vous serez éventuellement amené à adapter certaines de ces entités en fonction des spécifications du nouveau `Sprint#i+1`.

Attention : Pour éviter le partage de sources sous Visual Studio, la copie des fichiers (.h ou .cpp) se fera par copie au niveau du système de fichier.
Clic Droit sur le **projet à partir duquel** vous voulez **copier des sources**
« **Ouvrir le dossier dans l'Explorateur Windows** »
Sélectionnez les sources à copier clic Droit « **Copier** »
Clic Droit sur **le projet où vous voulez copier les sources**
« **Ouvrir le dossier dans l'Explorateur Windows** »
clic Droit « **Coller** » : les sources seront copiés dans les répertoires correspondants (Fichiers d'entête et Fichiers sources)

Développement par Sprint 2/2

1. **Dans le cas où votre programme ne fonctionne pas sur l'ensemble des fonctionnalités demandées**

   Dans le dossier de développement logiciel demandé,
   vous donnerez un rapport sur tous les Sprints que vous avez passé avec succès sur les JDT de Développement : aucune trace n'est demandée.
   Le jour de la recette, vous passerez votre recette sur le Sprint de plus niveau sur un JDT de recettes correspondant. En cas d'échec, votre enseignant contrôlera que vous passez bien pour ce Sprint les tests sur le JDT de Développement .

2. **Dans le cas où votre programme fonctionne sur l'ensemble des fonctionnalités demandées,**
   suivez les spécifications données dans le projet pour le dossier de développement

Labyrinthe de Développement – `inSmall`

```
30 4
#######D########################
+#++#++++++#####+++##########++++
#+###+#####++#######++###+##+#++
##############################

##############################
+#+########++++++####++#######+
+++++P++++++++#++++#+++++++++
##############################
```


Premier Sprint – Sprint#1 1/5

Après analyse, le premier Sprint proposé est le suivant : à partir de la `Solution`, créez un projet de nom `Sprint1` par « Ajouter» « Nouveau projet… ».

- **Analyse fonctionnelle** : A partir d'un fichier texte décrivant le labyrinthe (e.g., `inSmall.txt` sur COMMUN), lire et afficher le labyrinthe lu à partir de deux tableaux implémentés en mémoire dynamique (**`TableauMb`**).
- **Spécification** : Développez les composants (.h et .cpp) **`LabyrintheMb`** et **`TableauMb`** ainsi que le programme de test (`main`) correspondant à l'analyse fonctionnelle précédente.
- **Codage** : Codez **a)** les fonctions `initialiser` et `detruire` de `TableauMb` et **b)** les fonctions `initialiser`, `detruire`, et `afficherSp1` de `LabyrintheMb`. Spécialisez `ItemMb` en caractère.
- **Test** : Exécutez le programme de test sur les labyrinthes de développement donnés sur **`COMMUN`** pour le développement. Vérifiez que les résultats obtenus sont bien les résultats attendus : le labyrinthe affiché est bien le labyrinthe initial.

Résultat de référence – `outSmallSp1/inSmall`

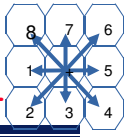
```
30 4
#######D########################
+#++#++++++#####+++##########++++
#+###+#####++#######++###+##+#++
##############################

##############################
+#+########++++++####++#######+
+++++P++++++++#++++#+++++++++
##############################
```

Deuxième Sprint – Sprint#2 2/5

Créez un nouveau projet de nom `Sprint2`. Recopiez les fichiers utiles de `Sprint1` et complétez le produit logiciel obtenu à la fin du premier Sprint.

- **Analyse fonctionnelle** : Donnez le labyrinthe visualisant les 10 premières cases explorées à partir de l'algorithme de recherche donné, puis la pile des positions.
- **Spécification** : Modifiez le composant **`LabyrintheMb`** et développez les composants (`.h et .cpp`.) **`DragonGame`** et **`IndexPositionMb`** ainsi que le programme de test (`main`) correspondant à l'analyse fonctionnelle précédente.
- **Codage** : Codez **a)** la fonction `afficherSp2` de **`LabyrintheMb`** qui affiche le labyrinthe en marquant `0` la 1ère case visitée, `1` la 2ème case visitée et ainsi de suite jusqu'à `9`ème case visitée puis le contenu de la pile, **b)** les trois fonctions `de DragonGame : initialiser, detruire et MissionDragonSp2` (exploration des dix premières cases) et **c)** la fonction `initialiser` de `IndexPosition3D`. Introduisez le type énuméré `Case` (cf. `Item.h`).
- **Test** : Exécutez le programme de test sur les labyrinthes de développement. Vérifiez que les résultats obtenus sont bien les résultats attendus : L'emplacement des 10 premières cases explorées dans le labyrinthe : de l'entrée (0) à la position du dragon (9).

Résultat de référence – `outSmallSp2/inSmall`




```
30 4
#######0########################
+#+++#78512#####+++##########++++
#+###9#6###34######++###+##+#++
##############################

##############################
+#+########++++++####++########+
+++++P++++++++#++++#+++++++++
##############################
```

(3,1,1) (6,1,1) (6,1,1) (7,1,1) (6,1,1)

Mouvements et contenu de la pile

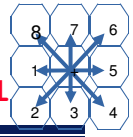
Case visitée	Mouvement autorisé	Pile d'Ariane
(7,0,1)	2 , 3, 4	(8,1,1) (7,1,1) (6,1,1)
(8,1,1)	1, 5	(9,1,1) (7,1,1) (7,1,1) (6,1,1)
(9,1,1)	4	(10,2,1) (7,1,1) (7,1,1) (6,1,1)
(10,2,1)	5	(11,2,1) (7,1,1)(7,1,1) (6,1,1)
(11,2,1)	-	(7,1,1) (7,1,1) (6,1,1)
(7,1,1)	1, 2	(6,2,1) (6,1,1) (7,1,1) (6,1,1)
(6,2,1)	7, 8	(5,1,1) (6,1,1) (6,1,1) (7,1,1) (6,1,1)
(5,1,1)	2, 5	(6,1,1) (4,2,1)(6,1,1) (6,1,1) (7,1,1) (6,1,1)
(6,1,1)	-	(4,2,1) (6,1,1) (6,1,1) (7,1,1) (6,1,1)
(4,2,1)	8	(3,1,1) (6,1,1) (6,1,1) (7,1,1) (6,1,1)

Troisième Sprint – Sprint#3 3/5

Créez un nouveau projet de nom `Sprint3`. Recopiez les fichiers utiles de `Sprint2` et complétez le produit logiciel obtenu à la fin de la deuxième itération

- **Analyse fonctionnelle** : Donnez le labyrinthe visualisant toutes les cases explorées à partir de l'algorithme de recherche donné.
- **Spécification** : Modifiez les composants (`.h` et `.cpp`) **`DragonGame`** et **`LabyrintheMb`** et développez le programme de test (`main`) correspondant à l'analyse fonctionnelle précédente.
- **Codage** : Codez **a)** la fonction `MissionDragonSp3` de `DragonGame` (exploration de toutes les cases suivant l'algorithme) et **b)** la fonction `afficherSp3` de **`LabyrintheMb`** qui affiche le labyrinthe en marquant "`V` " chaque case visitée.
- **Test** : Exécutez le programme de test sur les labyrinthes de développement. Vérifiez que les résultats obtenus sont bien les résultats attendus : l'emplacement de toutes les cases explorées dans le labyrinthe : de l'entrée à la position des plans du Monde.

Résultat de référence – `outSmallSp3/inSmall`




```
30 4
#######V########################
V#VV#VVVVV#####+++#########++++
#V##V#V###VV######+++#+##+#++
##############################

##############################
+#+#######VVVVVV####VV#######V
++++VVVVVVVVV#VVVV#VVVVVVVVV
##############################
```

2

Quatrième Sprint – Sprint#4

Créez un nouveau projet de nom `Sprint4`. Recopiez les fichiers utiles de `Sprint3` et complétez le produit logiciel obtenu à la troisième itération.

- **Analyse fonctionnelle** : Donnez le chemin connexe, allant de la position du dragon à l'entrée du labyrinthe, obtenu par l'algorithme de recherche après l'exploration des 20 premières cases.
- **Spécification** : Modifiez les composants (.h et .cpp) **`DragonGame`**, **`IndexPositionMb`** et **`LabyrintheMb`**, développez le programme de test (`main`) correspondant à l'analyse fonctionnelle précédente.
- **Codage** : codez **a)** la fonction `estConnexe` de `IndexPositionMb`, **b)** la fonction de `DragonGame` : `missionDragonSp4` (calcul du chemin connexe obtenu après l'exploration des 20 premières cases) et **(c)** la `afficherSp4` de **`LabyrintheMb`** qui affiche le labyrinthe en marquant "`C`" le chemin du Dragon à l'entrée du labyrinthe.
- **Test** : Exécutez le programme de test sur les labyrinthes de développement. Vérifiez que les résultats obtenus sont bien les résultats attendus : le chemin allant de la position du dragon à l'entrée du labyrinthe, après l'exploration des 20 premières cases.

Résultat de référence – `outSmallSp4/inSmall`

```
30 4
#######C########################
C#CC#C+CC+#########+++#########+++
#C##C#C###++#######++###+##+#++
##############################

##############################
+#+#########++++++###+++#######C
+++++P++++++++##++++#++++CCCCC
##############################
```


Algorithme de vérification de la connexité du chemin

```
Soit filAriane une liste des cases visitées et C la
nouvelle case visitée


Tant que !Vide(filAriane)  et !Connexe(fileAriane[0],C)
      supprimer (filAriane[0])
fTantque

Inserer(filAriane[0],C)
```

Cinquième Sprint – Sprint#5 5/5

Créez un nouveau projet « C++ géré » de nom `Sprint5`. Recopiez les fichiers de `Sprint4` et compléter le produit logiciel obtenu à la fin de la troisième itération.

- **Analyse fonctionnelle** : Détectez s'il existe un chemin connexe allant de la position des Plans du Monde à l'entrée du labyrinthe. Donnez le chemin connexe s'il existe sinon affichez le mécontentement du Dragon.
- **Spécification** : Modifiez les composants (.h et .cpp) **`DragonGame`** et **`LabyrintheMb`**, développez le programme de test (`main`) correspondant à l'analyse fonctionnelle précédente.
- **Codage** : codez **a)** la fonction `missionDragonSp5` de `DragonGame` (recherche et existence d'un chemin connexe des plans du Monde à l'entrée du labyrinthe) et **b)** la fonction `afficherSp5` de **`LabyrintheMb`** qui affiche le labyrinthe en marquant "`C`" le chemin des plans du monde à l'entrée du labyrinthe ou le mécontentement du dragon.
- **Test** : Exécutez le programme de test sur les labyrinthes de développement. Vérifiez que les résultats obtenus sont bien les résultats attendus : le chemin allant de la position des plans du Monde à l'entrée du labyrinthe ou le mécontentement du Dragon.

Résultat de référence – `outSmallSp5/inSmall`

```
30 4
#######C############################
C#CC#C+CC+#####+++########++++
#C##C#C###++#######++###+##+#++
##############################

##############################
+#+#######CCCCCC####CC#######C
+++++CCCCCCCCC##CCCC#CCCCCCCCC
##############################
```


Labyrinthe de Développement – `inNon`

```
30 4
#######D############################
+#++++++++##########++++++++++++
#+##+#+###++#########++###+##+#++
##############################

##############################
+#+##########++++++###++########+
++++#P++++++++#+++++#+++++++++
##############################
```


Résultat de référence – `outNonSp5/inNon`

```
30 4
##############################
##############################
##############################
##############################

##############################
##############################
#####P########################
##############################
```

Mécontentement du dragon

Résultat de référence – `outSmallSp5/inSmall`

```
C(5,2,2)->C(6,2,2)->C(7,2,2)->C(8,2,2)-
>C(9,2,2)->C(10,2,2)->C(10,1,2)->C(11,2,2)-
>C(11,1,2)->C(12,2,2)->C(13,1,2)->C(12,1,2)-
>C(13,2,2)->C(14,1,2)->C(15,1,2)->C(16,2,2)-
>C(17,2,2)->C(18,2,2)->C(19,2,2)->C(20,1,2)-
>C(21,2,2)->C(21,1,2)->C(22,2,2)->C(23,2,2)-
>C(24,2,2)->C(25,2,2)->C(26,2,2)->C(27,2,2)-
>C(28,2,2)->C(29,2,2)->C(29,1,2)->C(0,1,1)-
>C(1,2,1)->C(2,1,1)->C(3,1,1)->C(4,2,1)-
>C(5,1,1)->C(6,2,1)->C(7,1,1)->C(8,1,1)-
>C(7,0,1)->
```


Dossier de développement logiciel

❑ Respect des spécifications

Respect des spécifications données pour le dossier de développement logiciel

Respect du cahier des charges de présentation
Présence dans le dossier a) une page de garde indiquant le nom et le groupe des membres du binôme/trinôme, l'objet du dossier, b) une table des matières de l'ensemble du dossier (incluant les annexes) avec la pagination de toutes les rubriques, la pagination est continue du début à la fin du dossier
c) présentation de l'application, e) organisation des tests, f) bilan de validation des tests, g) bilan de projet. Brochez vos dossiers.

❑ Présentation de l'application

L'art de synthétiser le projet dans un format donné (1 page).
On devra trouver dans cette présentation son rôle fonctionnel (ce que fait l'application), ses entrées et ses sorties (au moins pour le Sprint validé).

Qualité de code (1/2)

❑ Lisibilité du code

Code parfaitement indenté (tabulation)
Ligne de code limitée à 80 colonnes

❑ Documentation

Cartouche du fichier source (son nom, son/ses auteurs et a minima sa date de création)
Des types structurés et de leurs champs
Des variables importantes

Des fonctions :
rôle de la fonction, des paramètres formels,
leur mode ([in], [out] et [in-out])**, rôle du paramètre de retour éventuel**

```
/*
 * Calcul du maximum de deux valeurs
 * [in]  x, 1ere valeur
 * [in] y, 2ème valeur
 * [out] mx, le maximum
 */
void max(int x, int y, int* mx);
```


Qualité de code (2/2)

❑ Absence de nombres magiques

Absence de littéraux, utilisation des constantes (pas d'utilisation de `#define`) motivée par des changements possibles des constantes considérées.

❑ Longueur des fonctions

Le `main` doit être court (à très court, quelques lignes).
Un `main` est à un niveau macroscopique (constitué d'appels de fonction). Un main trop long traduit un manque au niveau de l'analyse, des fonctions aurait dû être conçues.
Par exemple on n'y développe pas une interface (e.g., un menu).
Une fonction ne doit pas dépasser une vingtaine de lignes. Une fonction trop longue indique souvent un manque d'analyse fonctionnelle et s'accompagne de redondance : dans ce cas, on relit et on analyse le code en visant un niveau mascroscopique (de l'algorithme) pour introduire les fonctions (à coder) et qui seront à appeler (la longueur du code diminuera).

❑ Code redondant

Introduire la/les fonction(s) qui généralisent les traitements.

Les tests

Tests

D'une manière générale, vous devez préciser votre stratégie de test : comment sont organisés vos tests.
Un test comprend : un objectif (ce que l'on veut tester), un JDT (jeu de test), un résultat attendu (résultat de référence), un résultat (trace d'exécution de votre programme) et un bilan de validation (ce que valide le test).
Il faut rédiger cette partie.

Dans le rapport, on doit trouver une partie de « Bilan de validation des tests » où on résume tous les tests et ce qu'ils valident. But : vérifier votre compréhension des tests.

Des JDT « personnels » servent à améliorer la couverture des tests.
2 tests personnels sont demandés (cf. projet)

Bilan de projet

Retour d'expérience. Les difficultés rencontrées, ce qui est réussi, ce qui peut être amélioré.

Evaluation des projets – Barème indicatif

[n/15 pts] NS	Note de Sprint (du plus haut niveau `#n`) validé lors de la recette
[1 pt] MOD	Modulation d'un Sprint de niveau `n` validé relativement au source du `sprint#n+1` (si présent dans le dossier)
[1 pt] PA	Présentation de l'application
[1 pt] BV	Bilan de validation
[1 pt] BP	Bilan de projet
[2 pts] QC	Qualité du code
[-1 pt] PN	Pénalités de non respect des spécifications (cf. texte du projet) Ex. N° de groupe absent sur la page de garde, pagination absente..

Barème indicatif
NS [/15 points] Barème établi sur le Sprint de plus haut niveau validé à la recette

Sprint 5 : 15 points
Sprint 4 : 13 points
Sprint 3 : 09 points
Sprint 2 : 07 points
Sprint 1 : 03 points
Sprint 1 non atteint : 0

ATTENTION
0/20 si le dossier de développement logiciel n'est pas déposé
0/20 si le code n'est pas présent dans le dossier