Méthodes de développement

Gireg Desmeulles, Eric Maisel, Pierre De Loor

ENIB

January 29, 2016

Table des matières

Table des matières

Table des matières

Introduction

Rappel des notions de S1

S1

Que vous rappelez vous?

Objectifs du cours

Coder

- Mettre en application les notions de S1
- L'enibien Vs la machine!

Mener un projet

- Acquérir quelques outils pour définir, concevoir et réaliser un logiciel en partant de l'expression d'un besoin.

Déroulement

S1		Introduction
S2-S4	CTD	Elements techniques
S5-S7	CTD	Méthodes de développement
S7-S14	TP	Codage

Attention

Le cours de MDD est en cours d'évolution. Le plan du cours pourra évoluer au cours du semestre. Certains supports sont encore imparfaits. Il n'y a pas encore de document de synthèse.

Initiation

Il est impossible de tout traiter. Ce cours vous donne quelques éléments techniques pour comprendre les éléments méthodologiques abordés. Dans la suite du cursus ENIB vous approfondirez beaucoup de ces notions.

Travail à fournir

A la fin de chaque cours

Description du travail à fournir pour la fois suivante

Entre chaque cours

Relire le cours, essayer les éléments de codages indiqués

Au début de chaque cours

Etre capable de répondre à des questions qui montrent que le travail a été fait. Nous ferons attention aux cours trop rapprochés.

Binôme

Ne vous reposez pas sur votre binôme!

Evaluation

- QCM (contrôle continu)
- Analyse du besoin (TP + contrôle continu)
- Conception (TP + contrôle continu)
- Codage (TP)
- DS (DS)

Projet

Réalisation d'un jeu

- python
- programmation procédurale
- boucle de simulation
- barrière d'abstraction
- méthodes de développement : cahier des charges, conception...
- interface texte
- original

Exemple:

Casse briques, pac man, shoot them up, jeu de donjon, jeu d'aventure, zelda, course...

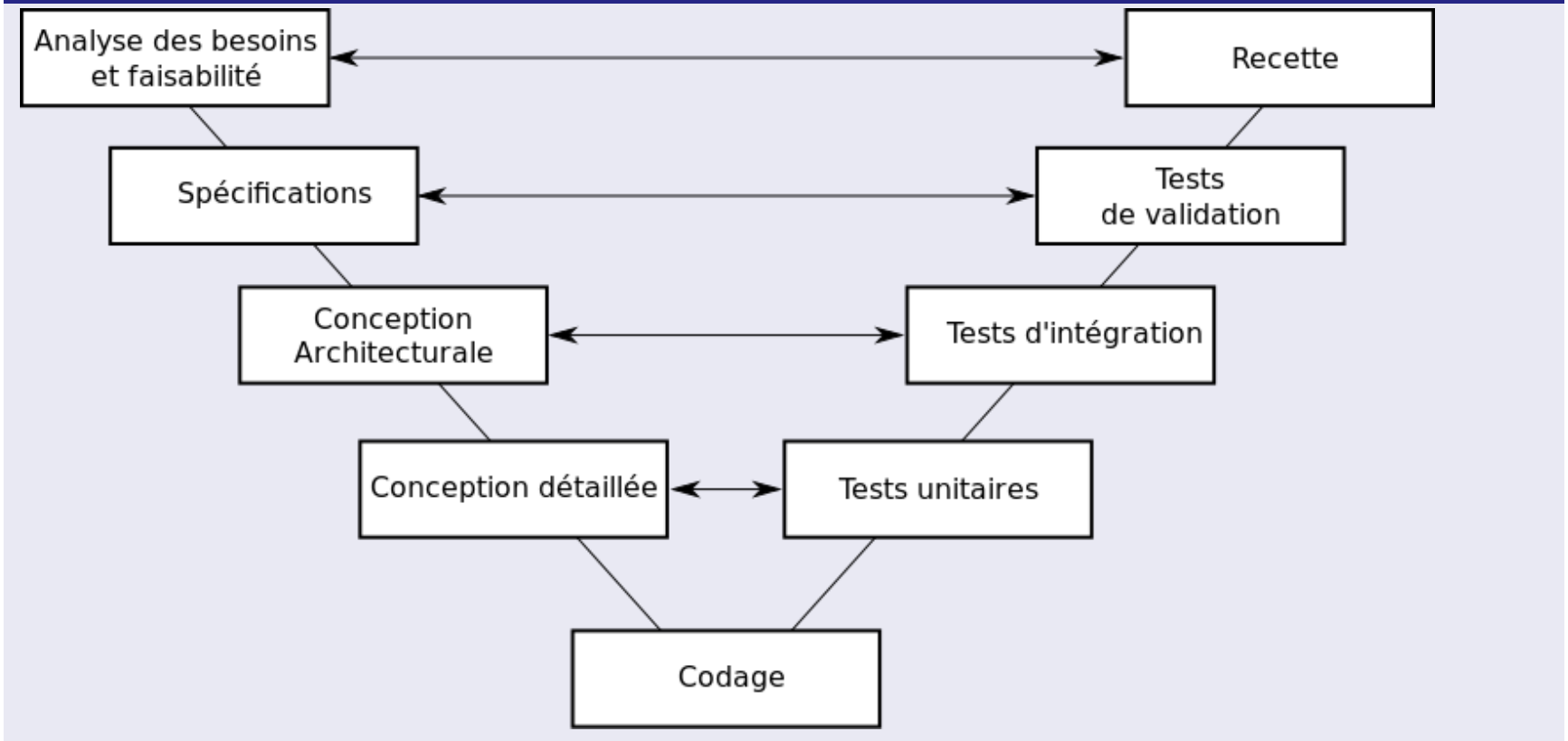
Un exemple de code

Un exemple de code:

A partir de cet exemple on peut faire beaucoup de choses!
Savoir lire avant d'écrire!

Cycle en V

De l'expression du besoin... à la livraison du logiciel.



Notation du projet

- Le résultat compte
- La façon d'y arriver est aussi importante
- Reprendre du code existant ne fonctionne pas

A faire

Pour la prochaine séance :

- Retrouver le cours de S1
- Relire Le Cours d'aujourd'hui.
- Exécuter l'exemple décrit en cours
  - ne fonctionne pas sous Windows.
  - dans un terminal : `$python main.py`

Environnement de développement

Systèmes d'exploitations (OS)

Un **système d'exploitation** est un logiciel qui fait fonctionner l'ordinateur et qui permet aux logiciels applicatifs de s'exécuter.

En salle de TP

Distribution Linux : CentOS

Si vous avez votre propre machine

Par rapport aux contraintes du semestre:

- Distributions Linux
- MacOS
- Windows + Linux dans une machine virtuelle
- Dual Boot

A vous de maîtriser votre outil pour travailler dans un terminal Unix

Systèmes de fichiers

Un **système de fichier** permet de gérer le stockage des fichiers et leur organisation dans les mémoires secondaires (disque dur, CD ou clé). Chaque OS mets en oeuvre différents systèmes de fichiers.

Quels systèmes?

- Distributions Linux – Ext ou Ext2
- MacOS – HFS
- Windows – NTFS
- Tout le monde – FAT32

Lorsqu'on change de système

- Perte des métadonnées
- Perte des noms
- Perte de fichiers

Interpréteur (OS)

Logiciel qui exécute un programme écrit dans un langage donné. L'interpréteur est à distinguer du compilateur qui lui transforme un code en programme exécutable ultérieurement. Un interpréteur compile le code dynamiquement à chaque nouvelle exécution.

En salle de TP

Python 2.x

Si vous avez votre propre machine

Attention aux conflits python 2.x et 3.x. Si vous pouvez éviter 3.x, c'est mieux.

Logiciel tiers

Un **logiciel** tiers est un logiciel qui peut être ajouté à un autre logiciel. Un logiciel tiers peut être un **composant logiciel** développé par un autre éditeur qu'on utilise pour ne pas re-développer certaines fonctionnalités nécessaire à notre application. Les modules python que nous n'avons pas développés nous même peuvent être vus comme des logiciels tiers (tkinter, pygame, matplotlib...)

Précaution

- module natif ou non?
- dépendance à un OS?
- licence?

Propriété logicielle

Un logiciel peut être **libre** ou **propriétaire**.

Licences propriétaires

- freeware (gratuit)
- shareware (droit de copier)
- shared source (droit de voir)
- ...

Licence libres

- GNU General Public License (contraintes contre les contraintes)
- Domaine public (70ans après la mort de l'auteur)
- BSD Licence (GPL "sans" contrainte)

Environnements de Développement Intégrés

Définition

Un environnement de développement intégré (IDE) contient : un éditeur de texte, un compilateur ou un interpréteur, un débogueur...
Exemples: Visual Studio, Eclipse, MonoDevelop

Avantages

- productivité,
- automatisation de la compilation, édition de liens...
- documentation

Inconvénients

- inadapté à l'apprentissage de l'informatique,
- pas toujours libre
- parfois lié à un langage

Environnements de développement intégrés

Idle

Idle est un environnement de développement(IDE) dédié à python. Nous ne l'utiliserons pas car :

- conflits avec certaines dépendances
- inutilisable en S3, car langage C
- ambiguïté IDE / langage
- manque de fonctionnalité

Editeurs de texte pour le codage

Editeurs de texte en salle de TP

- Emacs, Xemacs, GNU Emacs (geek)
- VI, VIM, nano (codeurs fous)
- Kate, Kedit, gedit (gens normaux)
- OpenOffice (ceux qui n'ont rien compris!)

Faire un choix pour gagner en productivité

- raccourcis clavier
- fonctionnalités : couleur, indentation, complétion
- encodage des fichiers
- fenêtrage

vi(m)

vi(m) est un outils très puissant qui fonctionne sur tous les linux et sans interface graphique. Il est cependant très peut conviviale et nécessite un apprentissage de raccourcis clavier.

Exemples de raccourcis

- i = passer en mode insertion
- /unechaine = recherche une chaine de caractère "unechaine"
- x = suppression de caractère
- :w = sauvegarder
- :q = quitter

Emacs

Emacs est très puissant, un peu plus conviviale que VI.

Exemples de raccourcis

- ctrl x ctrl c = fermer
- ctrl x ctrl s = sauvegarder
- ctrl x ctrl f = ouvrir un fichier
- ctrl x ctrl b = changer de buffer
- ctrl _ = annuler
- ctrl g = sortir de la commande en cours de saisi
- atl w = copier
- ctrl w = couper
- ctrl y = coller

Kate

Editeur conventionnel.

Exemples de raccourcis

- ctrl w = fermer
- ctrl s = sauvegarder
- ctrl f = rechercher
- ctrl c = copier
- ctrl z = annuler

Encodage

Les éditeurs sauvegardent les fichiers avec un système d'encodage des caractères. Il y a plusieurs normes. Ainsi quand on ouvre un fichier avec la mauvaise norme, certains caractères son mal interprétés. Il peut également y avoir des erreurs de caractères de fin de ligne qui sont invisibles mais perturbent l'exécution du programme.

type d'encodage

- latin-1
- **UTF-8**
- ISO 8859-1

Documentation / autoformation

Vous devrez apprendre à vous documenter et à vous former. Il existe plusieurs sources d'information :

- Documentation technique (Application Programming Interface)
- Manuel utilisateur / site officiel
- Tutorial
- Forum

Précautions

- dates
- version
- sources peu fiables

Rappel des commandes linux

A faire

Pour la prochaine séance :

- Relire Le Cours d'aujourd'hui.
- Choisissez un éditeur
- Configurez votre matériel si vous en avez
- Consultez la page wikipedia "logiciel"
- Reprendre les exemples de cours avec "votre" éditeur
- Reprendre le code de l'exemple donné au premier cours : échangez le terme "animat" par "curseur" dans les fichiers et nom de fichiers.

Représentation des données

Représentation des données

- Comment représenter les données décrites dans le cahier les charges?
- Quelques instructions de bases pour les manipuler ?

Types de base

Type	Exemple	Mot clé	Passage par
Booléen:	a=False	bool	valeur
Entier:	b=-12	int	valeur
Réel:	c= 1.2 ou d=-5.4687e-2	float	valeur
+			
Chaîne:	e="toto" ou f='toto'	str	valeur
N-uplet:	g=12, "toto"	tuple	valeur
+			
Listes:	h=[1,2,3]	list	référence
Dictionnaire:	i='a':4,'r':8	dict	référence

Chaînes de caractères

Définition : Une chaîne de caractères est une séquence non modifiable de caractères (voir cours de S1).
Quand modéliser une donnée par une chaîne de caractères?

"...le joueur sera décrit par son score, **son nom**, sa position..."

"...Lorsqu'on rentre dans une nouvelle map, la **description** de la pièce est affichée sous la carte..."

"...le joueur saisit la **réponse** au clavier ..."

Chaînes de caractères

Chaînes et caractères ASCII

Une chaîne est une **séquence** de caractères. On peut sélectionner un des 128 caractères de la table ASCII de plusieurs manière.

- normal : str= "**A**BC"
- en octal : str= "**101**BC"
- en hexadecimal : str = "**x41**BC"

Chaînes et opérations

Certaines fonctions permettent de manipuler les chaines de caractères

- taille : len(str)
- concaténation : str = "méthodes de" + " développement"
- index : print str[3]
- conversion : format(float(int("12")))
- ... voir string.py

Chaînes de caractères

Encodages

```
import encodings
for e in sorted(set(encodings.aliases.aliases.values())):
        print e
```

Unicode

```
str = u"\u2540"
print str
```

http://www.unicode.org/fr/charts/PDF/U2600.pdf

N-uplet

Définition : Un N-uplet une séquence non modifiable d'éléments (voir cours S1). On peut l'assimiler à une liste qu'on ne pourrait pas modifier.
Quand utiliser les N-uplet?

Pour simplifier l'écriture ou pour passer plusieurs paramètres ou valeur de retour en une seule fois.

Pour modéliser des données non modifiables du programme.

Pour modéliser des clés d'entrée de dictionnaire.

N-uplet

image

```
origine= 10 , 50 , 90
print "x=",origine[0],"y=",origine[1]
```

Listes

Définition : Une liste est une séquence variable d'éléments (voir cours S1).
Quand utiliser les listes ?

"...le personnage possédera un certain nombre d'objets dans son inventaire... "

"...une question sera choisie au hasard parmi tous les QCMs disponibles..."

"...il est possible d'afficher les dix meilleurs scores du jeu... "

"...les joueurs posent leurs pions sur une grille de 8 par 8 cases...."

Listes

Exemple

```
l=[]
l.append(1)
l.append(2)
l.append(3)
for i in l:
    print i
```

listes de listes

Matrice 2D

```
matrix = [[0 for x in xrange(2)] for x in xrange(3)]
```

donne $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

image

```
img=[]
img.append([' ','o',' '])
img.append(['-','|','-'])
img.append([' ','|','\'])
```

Dictionnaire

Définition : un dictionnaire python est un conteneur qui associe des données à une clé. Le dictionnaire appartient à la famille des tables de hachage. La clé est souvent une chaîne de caractères mais peut être une autre donnée hachable (entier, réel, N-uplet).

"...Le joueur est caractérisé par un nom, un score, une position..."

"...Pour chaque question, on propose 3 réponses parmi lesquelles la bonne réponse..."

Dictionnaire

```
dic = {'prenom': 'Jean', 'nom': 'Dupond'}

for cle in dic:
    print cle, ":", dic[cle]

dic['tel']='0298056600'

if 'tel' in dic:
    print 'Téléphone:', dic['tel']

del dic['mail']
```

Arbres

Définition : un arbre est une structure de données récursive tel que chaque donnée, sauf la racine, possède un unique parent. Un arbre est un graphe particulier. Dans la pratique, non construirons des arbres avec de dictionnaires ou des listes.

"... Le jeu se compose de joueurs et d'un plateau. Chaque joueur est décrit par un nom, une position, une vitesse. Le plateau se décompose en cases, chaque case contient un mur ou une liste d'objets..."

Arbre

Représentez cet arbre de données

```
def initGame():
  global gameData
  img=[]
  img.append(['+','+','+','+','+'])
  img.append(['+',' ',' ',' ','+'])
  img.append(['+',' ',' ',' ','+'])
  img.append(['+',' ',' ',' ','+'])
  img.append(['+','+','+','+','+'])
  gameMap={'size':5,'matrix':img}

  monsters={(1,1):'troll',(3,2):'goblin'}

  items=['potion','sword','bow']
  player={'name':bob,'score':0,'bag':items,'position':(2,2)}

  gameData={'map':gameMap,'player':player,'monsters':monsters}
```

Graphes

Définition : un graphe est une collection de données mises en relation entre elles. Un graphe contient des noeuds (les données) et des liens (les relations binaires entre les données). Dans la pratique, non construirons des arbres avec des dictionnaires.

"...Le donjon contient plusieurs salles reliées par des portes..."

"...Le jeux reprend le principe des livres dont vous êtes le héros..."

Graphes

Exemple

```
p1={'nom':'cuisine','portes':[]}
p2={'nom':'salon','portes':[]}
p3={'nom':'couloir','portes':[]}
p4={'nom':'chambre','portes':[]}
p1['portes'].append(p2)
p1['portes'].append(p3)
p2['portes'].append(p1)
p2['portes'].append(p3)
p3['portes'].append(p2)
p3['portes'].append(p1)
p3['portes'].append(p4)
p4['portes'].append(p3)
```

Dessinez le!

Référence/Valeur

Porté des variables

- locale
- globale

Passage...

- ...par valeur
- ...par référence

A faire

Pour la prochaine séance :

- Relisez le cours d'aujourd'hui.
- Testez les exemples donnés en cours.
- Représentez sous forme d'un graphe le réseau routier reliant : Saint Renan, Brest, Plouzané, Plougastel, Quimper
- Documentez vous sur la fonction split() qui permet d'agir sur des chaînes. Comprenez la et testez la

Développement procédural

Programmation descendante

Décomposition

- Découpage d'un programme en fonctions/procédures
- Décomposition des problèmes (réductionnisme)
- Factorisation du code

Ecrire une fonction

1. Prototype
2. Commentaire
3. Structures de contrôle (for, if...)
4. Appel des sous fonctions

Arbre d'appel

Code

```
def f1():
  f2()
  f3()
def f2():
  pass
def f3():
  f2()
def main():
  f1()
def f5():
 pass
main()
```

Arbre d'appel

Redondance

Redondance du code

```
def movePlayer(player,dt):
  print ("Le joueur se déplace")
  vxp,vyp=player["speed"]
  xp,yp=player["position"]
  xp=xp+vxp*dt
  yp=yp+vyp*dt
  player["position"]=xp,yp

def moveMonster(monster,dt):
  msg=monster["name"]+" se déplace."
  print (msg)
  vxm,vym=monster["speed"]
  xm,ym=monster["position"]
  xm=xm+vxm*dt
  ym=ym+vym*dt
  monster["position"]=xm,ym
```

Redondance

Redondance du code

```
def movePlayer(player,dt):
  print ("Le joueur se déplace")
  vxp,vyp=player["speed"]
  xp,yp=player["position"]
  xp=xp+vxp*dt
  yp=yp+vyp*dt
  player["position"]=xp,yp

def moveMonster(monster,dt):
  msg=monster["name"]+" se déplace."
  print (msg)
  vxm,vym=monster["speed"]
  xm,ym=monster["position"]
  xm=xm+vxm*dt
  ym=ym+vym*dt
  monster["position"]=xm,ym
```

Code factorisé

```
def movePlayer(player,dt):
  print ("Le joueur se déplace")
  move(player,dt)

def moveMonster(monster,dt):
  msg=monster["name"]+" se déplace."
  print (msg)
  move(monster,dt)

def move(a,dt):
  vx,vy=a["speed"]
  x,y=a["position"]
  x=x+vx*dt
  y=y+vy*dt
  a["position"]=x,y
```

Redondance

Intérêt

- Moins de code à écrire
- Moins de bugs à corriger
- Moins de code à maintenir

Le bon fainéant

- Une fonction d'une seul ligne, c'est possible!
- Plein de fonctions différentes, ce n'est pas grave!

Le mauvais fainéant

- Si on ne peut pas voir toute la fonction sans "scroller" c'est suspect!
- Programmation procédurale et non séquentielle !

Bugs

Personne ne code sans erreur!

- Fautes de frappe
- Fuites mémoire
- Erreurs de contrôle de boucle
- Erreurs de condition d'arrêt (boucle infinie)
- Divisions par 0
- Dépassements de capacité
- Erreurs d'initialisation
- Erreurs dans le passage des paramètres

Tests : branche ascendante du cycle en V

Tests unitaires

Sur partie d'un programme pour valider les fonctions indépendamment.

Tests d'integration

Lorsqu'on associe des parties de code déjà testées indépendamment.

Tests de validation

On teste l'adéquation entre les spécifications du document de conception et le logiciel réalisé.

Recette

Acceptation du logiciel par la maîtrise d'oeuvre

Tests de non régression

Permet de rejouer tous les tests à la suite d'un correctif ou d'une évolution.

Tester un module Python

Code

```
def movePlayer(player,dt):
  print ("Le joueur se déplace)
  move(player,dt)

def moveMonster(monster,dt):
  msg=monster["name"]+" se déplace."
  print (msg)
  move(monster,dt)

def move(a):
  vx,vy=a["speed"]
  x,y=a["position"]
  x=x+vx*dt
  y=y+vy*dt
  a["position"]=x,y
```

Test

```
if __name__ == '__main__':
  #test move
  a={"speed":(6,2),"position":(3,4)}
  move(a,0.5)
  if(a[position]!=(6,5)):
    print("move() test error")

  #test movePlayer
  movePlayer(a,0.5)
  if(a[position]!=(6,5)):
    print("movePlayer() error")

  #test moveMonster
  a["name"]="bob"
  moveMonster(m,0.5)
  if(m[position]!=(6,5)):
    print("moveMonster() error")
```

Découpage en module

Intérêt

- Lisibilité
- Modularité
- Réutilisabilité

A éviter :

- Découpage aléatoire
- Dépendances circulaires

Graphe de dépendance

Barrière d'abstraction

Principe : La **barrière d'abstraction** permet de cacher dans un module les détails d'implémentation des services rendus par ce module. Nous nous appuierons sur ce principe pour concevoir le découpage en modules de nos logiciels.

Abstraction

- Pour utiliser une fonction, il n'y a pas besoin de savoir comment elle est codée. Il suffit de connaître les spécifications qui fournissent une définition abstraite du service rendue.
- Pour manipuler des données complexes à l'aide de fonctions, il n'est pas toujours nécessaire de savoir comment ces données sont structurées.

Mise en oeuvre de la barrière d'abstraction

Un module par type

Regrouper dans un module toutes les fonctions qui manipulent la même donnée.

`cityInfo.py`

Constructeur

Définir une fonction pour créer la donnée.

```
def create(name='',population=0):
  cityInfo={'name':name,'population'=population}
  return cityInfo
```

Mise en oeuvre de la barrière d'abstraction

Accesseurs

Proposer des accesseurs pour retrouver des informations décrites par la donnée. On utilise le mot clé **get**.

```
def getName(cityInfo):
  return cityInfo['name']
def getPopulation(cityInfo):
  return cityInfo['population']
```

Mutateurs

Proposer des mutateurs pour modifier la donnée. On utilise le mot clé **set**.

```
def setName(cityInfo, name):
  cityInfo['name']=name
def setPopulation(cityInfo,number):
  cityInfo['population']=number
```

Mise en oeuvre de la barrière d'abstraction

Opérations

Chaque fonction permet de réaliser une opération sur les données

```
def show(cityInfo):
  msg= "La ville de"+ cityInfo['name']+" comporte "
  msg=msg+ cityInfo['population']+ " habitants"
  print(msg)
```

Type abstrait de donnée

- le nom du module correspond à la donnée
- le module fournit les opérations pour cette donnée.
- s'interdire de modifier directement une donnée sans utiliser les opérations fournies
- décomposer les fonctions au besoin pour que chaque portion de code soit dans le bon module.

A faire

Pour la prochaine séance :

- Relisez le cours d'aujourd'hui.
- Structurez votre code de test de la séance précédente (sur les villes) de façon à y ajouter une barrière d'abstraction.

Boucle de simulation

5 Boucle de Simulation
- Principe
- Résolution d'équations différentielles
- Machines à états

Contexte

- Nous nous plaçons dans le cadre de la réalisation d'un jeu qui fonctionne en plus ou moins en temps réel.
- **Simulation:** L'état du jeu évolue avec le temps qui passe.
- **Gestion des événements:** Le logiciel doit réagir au action du joueur.
- **Gestion de l'affichage**: L'utilisateur doit voir ce qui se passe dans le jeux.

Réalisation :

Nous avons besoin de mettre en oeuvre une boucle de simulation

Principe

Définition : Une **boucle de simulation** est une boucle infinie qui permet de réaliser les différentes actions nécessaires à la mise en oeuvre d'un logiciel de simulation.

Paramètres de la boucle :

- pas de simulation,
- fréquence d'affichage
- temps de latence/réponse

```
import time
timeStep=0.1 #pas=frequence=latence=timeStep
while True:
  t0=time.time()
  interact()
  live(timeStep)
  show()
  time.sleep(timeStep - (time.time()-t0))
```

Résolution d'équations différentielles

Il est possible d'utiliser une boucle de simulation pour réaliser à chaque tour de boucle, un calcul d'intégration d'équations différentielles sur un pas de temps, en utilisant une méthode de résolution.
Quand utiliser une resolution d'équations différentielles?

"...la balle se déplace à une certaine vitesse..."

"...les boulets de canon ont une vitesse initiale et sont soumis à la gravité..."

Résolution d'équations différentielles

Intégration : méthode d'Euler

$$\begin{cases} \frac{dx}{dt} = v_x \\ \frac{dy}{dt} = v_y \end{cases}$$

$$x_t = x_{t-1} + v_x * dt$$

$$y_t = y_{t-1} + v_y * dt$$

Code

```
while True:
  position[0]=position[0] + (dt * vitesse[0])
  position[1]=position[1] + (dt * vitesse[1])
```

Principe

Définition : Une machine à états finis peut être vue comme un graphe auquel on associe les sommets à des états et les liens à des transitions. De plus, la machine à états possède un état courant.
Quand utiliser une machine à état?

"...Les monstres se déplacent aléatoirement. Dès qu'ils repèrent le joueur, ils se dirigent dans sa direction..."

"...Le jeux reprend le principe des livres dont vous êtes le héros..."

Code

solution minimaliste:

```
state="a"
t=0

#boucle de simulation
while state!="exit":
  if (state=="a"):
       if t > 10 :
           state ="b"
  elif (state=="b"):
      state ="c"
  elif (state=="c"):
      if t > 100 :
         state ="exit"
  print state
  t=t+1
print ("fin de boucle")
```

A faire

Pour la prochaine séance :

- Relisez le cours d'aujourd'hui.
- Testez les exemples donnés en cours.
- Reprenez vos programmes de test sur les graphes de villes en utilisant le module stateMachine.py pour vous déplacer d'un ville à l'autre.

Détection des collisions

Détection des collisions

Définition : La détection des collisions est un mécanisme permettant de repérer des déplacements interdits ou nécessitant un traitement particulier, au cours de la simulation d'objets en mouvement.

"...la balle rebondit sur les murs..."

"...le joueur se déplace dans un labyrinthe. Il doit éviter les monstres qui s'y promènent..."

Dans la boucle de simulation

La détection des collisions intervient dans la partie "simulation" de la boucle de simulation.

- **Avant:** pour interdire un déplacement qui provoque une collision.
- **Après:** pour détecter puis gérer les objets en collisions pour laisser le système dans un état correct.

On peut effectuer la détection et le traitement des collisions à chaque petit déplacement ou une seule fois par pas de simulation.

Réalisation :

- fonction de détections de collisions
- représentation des "obstacles"

Mise en équations d'un rectangle

```
def move(dt,balle,xMax,yMax):
  x0,y0=balle['position']
  vx,vy=balle['vitesse']
  x1=x0+dt*vx
  y1=y0+dt*vy
  balle['position'] = x1,y1

  collide(balle,xMax,yMax)
```

Limitation

Environnement très simple

```
def collide(balle,xMax,yMax):
  x,y=balle['position']
  vx,vy=balle['vitesse']
  if(x<0):
    x=0
    vx=-vx
  if(y<0):
    y=0
    vy=-vy
  if(x>xMax):
    x=xMax
    vx=-vx
  if(y>yMax):
    y=yMax
    vy=-vy
  balle['position']=x,y
  balle['vitesse']=vx,vy
```

Collection d'obstacles

```
def move(dt, player, fireZones):
  x0,y0=player['position']
  vx,vy=player['vitesse']
  x1=x0+dt*vx
  y1=y0+dt*vy
  player['position'] = x1,y1

  collide(player, fireZones)
```

```
def collide(player, fireZones):
  x,y=player['position']
  for f in fireZones:
    #detection
    if fireZone.isInside(f,x,y):
      #gestion
      player['health']=player['health']-1
```

Limitation

On teste tout à chaque fois

Grille 2D

Principe: L'affichage et la simulation se basent sur une représentation partagée du "monde" simulé. Cette représentation peut être une grille dans laquelle on insère les différents éléments du monde.

Grille2D

```
world=[]
world.append(['+','-','-','-','-','-','+'])
world.append(['|',' ',' ',' ',' ',' ','|'])
world.append(['+','-','+',' ',' ',' ','|'])
world.append([' ',' ','|',' ',' ',' ','|'])
world.append([' ',' ','+','-','-','-','+'])
```

Grille 2D

Collisions

```
#player.py
def move(dx,dy,player,w):
  x,y=player['position']
  x=x+dt
  y=y+dy
  if (world.isValide(w,x,y)):
    player['position']=x,y

#world.py
def isValid(w,x,y):
  if (w[int(y)][int(x)]==' '):
    return True
  else:
    return False
```

Complexité

Attention :

- temps de calcul
- très compliqué lorsque que plusieurs éléments entrent en collision en même temps
- manque de robustesse
- manque de réalisme

Solution :

Soyez astucieux! La barrière d'abstraction peut vous aider.

A faire

Pour la prochaine séance :

- Relisez le cours d'aujourd'hui.
- Testez les exemples donnés en cours.
- Reprenez l'exemple de base en utilisant le fond d'écran pour les collisions

Evolutivité, persistance, XML

Evolutivité et persistance

Principe : Le fait de pouvoir faire évoluer un jeu en ajoutant facilement de nouveaux niveaux, scenarios, etc.. concours au caractère **évolutif** du logiciel.
Principe : La notion de **persistance des données** se réfère aux mécanismes de sauvegarde et de restauration des données. Une donnée persistante pourra se retrouver d'une exécution à l'autre du logiciel.

"...le menu "meilleurs scores" permet d'afficher les 10 meilleurs score de tous les temps..."

"...Il est possible d'ajouter facilement de nouvelles questions pour améliorer l'intérêt du jeu..."

Mise en oeuvre

La mise en oeuvre de l'évolutivité et de la persistance des données nécessite de lire et d'écrire dans des fichiers.

Type de fichiers

- Texte
- XML
- ...

Avantages

- séparation du code et des données
- modification facile des paramètres par l'utilisateur
- possibilité de créer un éditeur de niveaux

Accès aux fichiers

Fonction python

- **open()**: ouvre un fichier présent dans le répertoire courant
- **chdir()**: change le répertoire courant

Opérations des fichiers

- **read()**: permet de lire le contenu du fichier
- **readline()**: lit la prochaine ligne.
- **write()**: permet d'écrire dans le fichier
- **close()**: ferme un fichier

opérations et objet

monFichier.operation()

"r","w","a"

Lecture

```
myFile=open("file.txt","r")
str=myFile.read()
print(str.split("\n"))
myFile.close()
```

Écriture

```
myFile=open("file.txt","w")
myFile.write("écrase tout")
myFile.close()

myFile=open("file.txt","a")
myFile.write("\n ajoute en fin de fichier")
myFile.close()
```

XML

Besoin

Lorsque le nombre des données sauvegardées dans un fichier devient conséquent, il devient nécessaire structurer les fichiers de données.

Une solution

Le langage XML (eXtended Markup Langage) est un langage de description utilisant des balises pour structurer des fichiers texte.

avantages

- norme très utilisée
- modules déjà existants
- extensible

Minidom

Exemple

utilisation_dom.py

A faire

Pour la prochaine séance :

- Relisez le cours d'aujourd'hui.
- Testez les exemples donnés en cours.
- Essayez de sauvegarder/charger un graphe de ville avec minidom.
- Commencez à imaginer quel jeu vous voudriez faire.

Expression et analyse du besoin

Objectifs du cours

Elements techniques

- Mettre en application les notions de S1
- L'enibien Vs la machine!

Mener un projet

- Acquérir quelques outils pour définir, concevoir et réaliser un logiciel en partant de l'expression d'un besoin.

Projet MDD: réalisation d'un jeu

Réfléchir avant de coder!

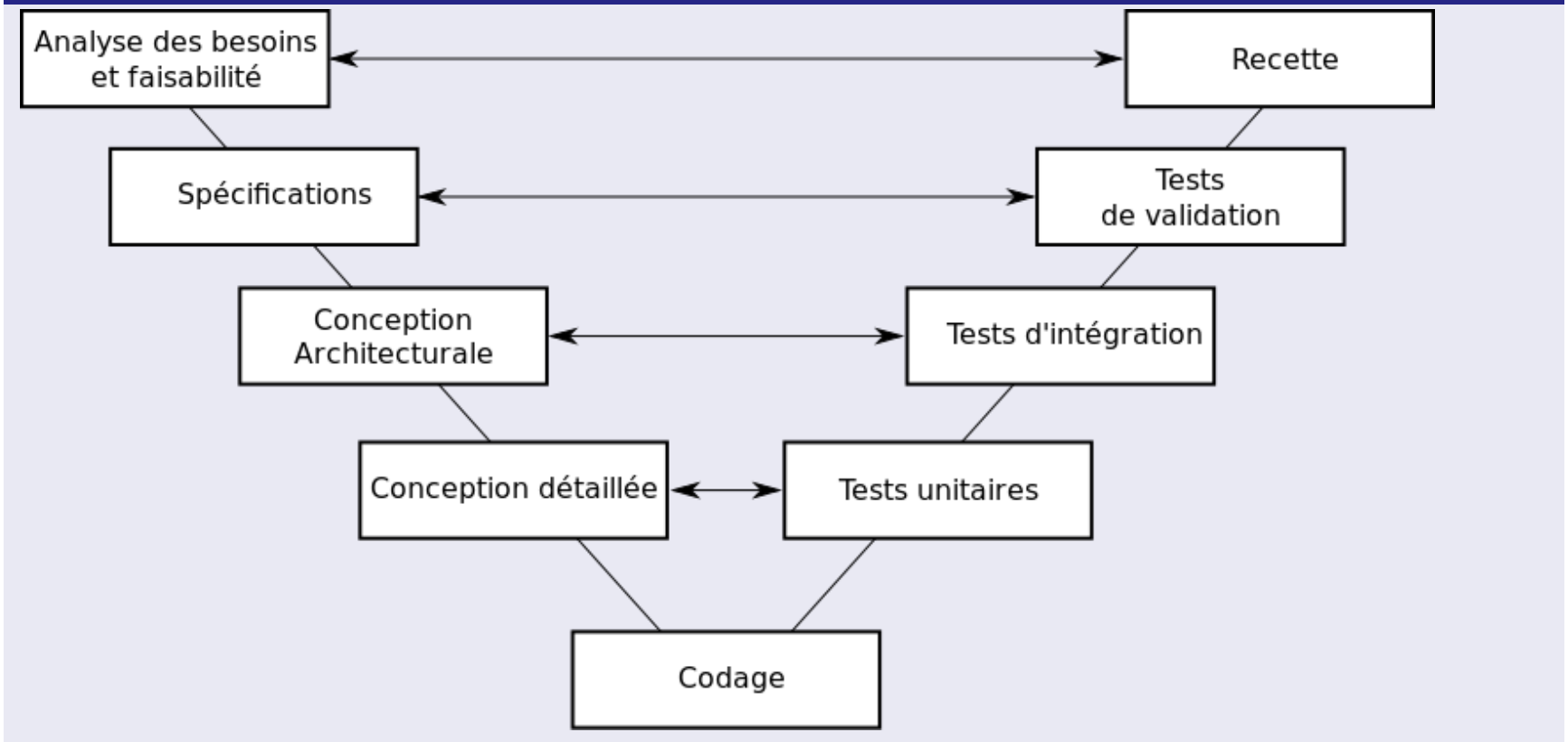
- projet inintéressant
- projet irréalisable
- projet inachevé
- projet incohérent
- projet inadapté

Avant de coder

1. expression du besoin
2. analyse du besoin
3. conception

Cycle en V

De l'expression du besoin... à la livraison du logiciel.



Acteurs du projet

Maître d'ouvrage

- Définition : client qui demande la réalisation
- Synonyme : client, product owner

Maître d'oeuvre

- Définition : fournisseur qui réalise le projet
- Synonyme: Développeur, fournisseur, prestataire, concepteur

Le besoin

La réussite d'un projet passe impérativement par la définition écrite, détaillée, précise, exhaustive et évaluable du besoin et des contraintes. Les phases d'expression et d'analyse du besoin permettent de décrire les fonctionnalités du logiciel et les contraintes sous lesquelles celui-ci doit être réalisé.

Expression du besoin

Les besoins sont exprimés par le maître d'ouvrage dans le cahier des charges et rédigés en langage naturel. L'expression du besoin est souvent associée à la description d'un contexte et de contraintes.

Analyse du besoin

L'analyse du besoin est réalisé par le maitre d'oeuvre en collaboration avec le maître d'ouvrage. Il s'agit d'identifier les fonctionnalités du produit à réaliser, sa faisabilité et sont coût. Pour être réellement utilisable, l'analyse doit également fournir des critères de validation et de qualité.

Besoin et fonction

Besoin

- Définition : Situation de manque ou prise de conscience d'un manque.
- Format : langage naturel, dessin, schéma...

Fonctions

- Définition : Eléments de rendu de service de l'application qui permettent de répondre au besoin exprimé.
- Synonyme : fonctionnalités
- Format : phrases à l'infinitif, sujet=système, compléments=acteurs en interaction avec le système.

Point de vue utilisateur

Il ne faut surtout pas décrire COMMENT le logiciel remplira ses fonctions.

Besoin et fonction

Exemples

- Crayon
  - Fonction : déposer de l'encre sur un papier
  - Besoin : communiquer
- Tondeuse
  - Besoin : entretenir le jardin
  - Fonction : diminuer la hauteur de l'herbe
- Projet de MDD
  - Besoin : Vous avez le concept d'un nouveau jeu et vous voulez permettre au prof et à vos amis d'y jouer.
  - Fonction : permettre à l'utilisateur de jouer à ce jeu.

Cahier de charges

Contenu

En MDD, le cahier des charges comportera l'expression et l'analyse du besoin.

Document contractuel

Dans une relation client / prestataire, le cahier des charges est un document contractuel. C'est en s'appuyant sur cahier des charges qu'on pourra déterminer si les objectifs ont été remplis.

Normes...

- Cahier des Charges Fonctionnel : AFNOR
- Use Cases : UML
- Récits d'utilisation : méthodes agiles
- ... Il faut s'adapter au sujet traité.

Cahier des charges en MDD

Fondation du projet

- Tout repose sur ce document.
- Mettez vous dans la peau d'un client. Imaginez qu'un autre binôme utilise votre cahier des charges.
- Le CdC vous engage. Certaines fonctionnalités peuvent être optionnelles.

Plan imposé en 4 parties

- Objectifs
- Expression du besoin
- Analyse du besoin
- Livrables

Cahier des charges en MDD

Page de garde

Mettre un **titre**, le **logo** de l'ENIB et une **illustration**. Utiliser un **cartouche** normalisé pour tous les documents du projet : type du document, nom du projet, commentaire, auteur, version, date.

Objectifs

- **Description générale**: 1 à 2 phrases maximum disant ce qu'est le document et le projet.
- **Contexte** : précise le but et le contexte du projet. Cela apporte un éclairage qui permet de mieux comprendre le besoin qui sera exprimé

Cahier des charges en MDD

Expression du besoin

- **Règles du jeu**: Règles complètes, précises et exhaustives
- **L'interface utilisateur**: Proposition de vues et description des moyens d'action.
- **Manuel utilisateur**: Première version du manuel
- **Contraintes techniques**: Au moins celles énoncées pour MDD
- **Scenario d'utilisation**:
  - schéma (non formalisé)
  - éventuellement plusieurs.
  - enchainement d'actions.
  - phrases à l'infinitif, le sujet est l'utilisateur (sauf exception explicitée)

Cahier des charges en MDD

Analyse du besoin

- **Fonctionnalité**: c'est le plus important!
- **Critère de validité et de qualité**: essentiel dans la relation client/prestataire

Cahier des charges en MDD

Livrables

- **Echéancier**: Calendrier indiquant quand seront livrés les livrables.
- **Documents**: Brève description de chaque documents.
- **Prototypes**: Description des prototypes et des fonctionnalités qu'ils devront posséder.

A faire

Pour la prochaine séance :

- Mettez vous en binôme
- Trouver un concept de jeu
- Commencer la rédaction d'un cahier de charges en s'appuyant sur l'exemple de cours.

Conception

10 Conception

Le document de conception

Le document de conception se base sur le cahier des charges il décrit la conception du logiciel à réaliser.

Avant le codage

- définition/choix des principes généraux
- description du code à réaliser
- planification du travail

Pendant le codage

Document de référence pour:

- coder sans se poser de question d'ordre général
- débugger
- vérifier l'avancement du projet
- travailler à plusieurs

Le document de conception en MDD

Rédaction

- Ecriture simultanée des différentes parties
- Cohérence d'une parie à l'autre

Plan imposé en 4 parties

- Rappel du CdC
- Principes des solutions techniques
- Analyse
- Description des fonctions
- Calendrier et suivi de développement

Le document de conception en MDD

Page de garde

Mettre un **titre**, le **logo** de l'ENIB et une **illustration**. Utiliser un **cartouche** normalisé pour tous les documents du projet : type du document, nom du projet, commentaire, auteur, version, date.

Rappel du cahier des charges

Le point de départ de la conception sont les fonctionnalités et les prototypes à implémenter.

Le document de conception en MDD

Principes de solutions techniques

Expliquer les principes retenus (en s'appuyant sur le cours).

Analyse

- Analyse noms-verbes (pour trouver les types de donnée)
- Type de données
- Dépendance des modules
- Analyse descendante (Arbres d'appel des fonctions)

Le document de conception en MDD

Description des fonctions

Spécifications des fonctions par module

Calendrier et suivi du développement

Partie utilisée pendant les développement par les développeurs

A faire

Pour la prochaine séance :

- Commencer la rédaction d'un document de conception en vous basant sur votre cahier des charges.