MVC et hamsters...

Le patron de conception "Modèle - Vue - Contrôleur" est destiné à faciliter le développement d'interfaces graphiques.

Une Interface graphique est constituée essentiellement de deux modules :

- Le "*front-end*" (la "devanture"), qui est la partie du programme visible pour l'utilisateur et avec laquelle l'utilisateur peut interagir à l'aide de menus, de boutons et de formulaires...
- Le "*back-end*" (l'"arrière-boutique"), qui correspond aux rouages invisible à l'utilisateur, permettant au programme de réaliser la tâche pour laquelle il a été conçu.

Pour développer un tel programme, on le divise généralement en trois modules appelés respectivement:

- le Modèle
- la Vue
- le Contrôleur

Le Modèle

Le modèle est la partie du programme qui manipule et met à jour les informations qui doivent être conservées d'une session à l'autre. Il s'agit de l'ensemble des variables et objets qui sont créés et mis à jour par l'utilisateur lorsqu'il interagit avec le programme.

La Vue

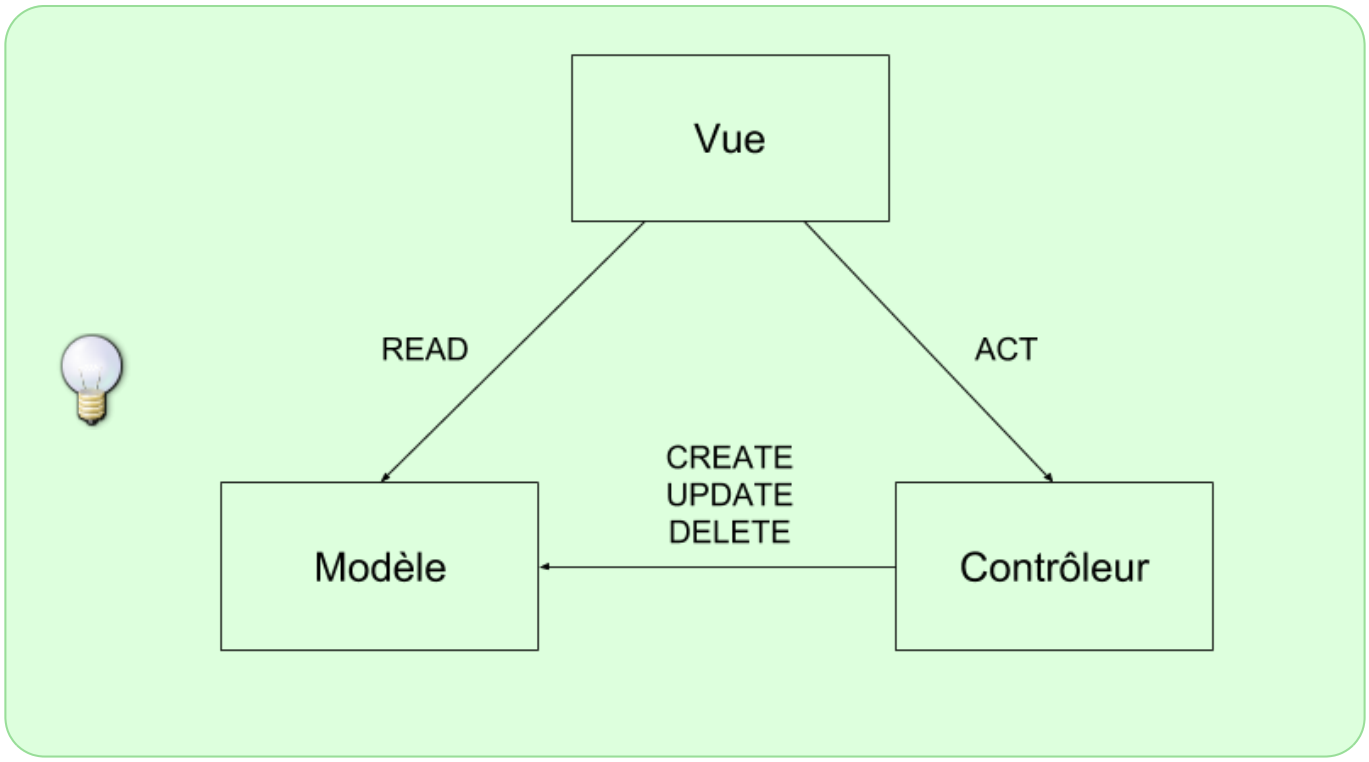
La Vue est la partie du programme qui gère la mise en page, la disposition des informations, des boutons et des formulaires, l'organisation et la visibilité des différentes fenêtres du programme s'il y en a.

- La Vue fait appel au Contrôleur à chaque fois que l'utilisateur effectue une action (saisie d'information, pointage de souris, sélection de menu, activation d'un bouton etc...)
- La Vue fait appel au Modèle pour afficher en permanence un contenu actualisé par les actions de l'utilisateur.

Le Contrôleur

Le contrôleur est la partie du programme qui gère les actions de l'utilisateur. Chacune des actions proposées dans la vue est implémentée dans le contrôleur sous la forme d'une fonction.

- Le contrôleur fait appel au Modèle lorsque l'action modifie les variables et objets manipulés par le programme.



Mise en œuvre

Dans ce TD, nous développons une application permettant de gérer une petite animalerie. Le programme gère un cheptel de rongeurs (tamias, hamsters, etc..) qui vivent dans une cage.

- La cage est constituée d'une litière, ainsi que d'une mangeoire, un nid et une roue pour faire de l'exercice. La mangeoire, le nid et la roue ne peuvent accueillir qu'un seul animal. La litière peut accueillir plusieurs animaux.
- Les animaux passent par différents états au cours de la journée:
  - Lorsqu'ils se réveillent, ils sont "affamés" et doivent donc être placés sur la mangeoire pour être nourris.
  - Une fois qu'ils ont mangé, ils sont "repus" et ont besoin d'exercice. Ils doivent donc être placés sur la roue.
  - Après avoir fait du sport, ils sont "fatigués" et ont besoin de dormir. Ils doivent donc être mis dans le nid pour se reposer.
  - et ainsi de suite...

L'état de notre animalerie est décrit à l'aide de deux fichiers `json`:

- animal.json
- équipement.json

Nous allons procéder par essai/erreur pour développer notre programme. Nous développerons, dans l'ordre:

1. Le Modèle
2. Le Contrôleur
3. La Vue

1. Le Modèle

Commençons par le Modèle. Le Modèle est la seule partir du programme autorisée à manipuler directement les données contenues dans les tables. Il permet essentiellement de réaliser des opérations de lecture et d'écriture dans les tableaux de données, qui doivent être mis à jour à chaque action de l'utilisateur.

Nous disposons pour l'instant uniquement de deux fichiers :

- le fichier animal.json contient l'état de cinq animaux, décrits par leur identifiant (ici un nom), leur RACE, leur TYPE, leur ÉTAT et leur `LIEU`.

Voici en clair l'état initial des animaux:

Animal	**RACE**	**TYPE**	**ÉTAT**	**LIEU**
Tic	tamia	rongeur	affamé	litière
Tac	tamia	rongeur	affamé	litière
Patrick	hamster	rongeur	affamé	litière
Totoro	ili pika	rongeur	repus	mangeoire
Pocahontas	opossum	marsupial	endormi	nid

- le fichier équipement.json contient l'identifiant et l'état des différents équipements.

Voici en clair l'état initial de l'équipement:

Équipement	**DISPONIBILITÉ**
litière	libre
mangeoire	occupé
roue	libre
nid	occupé

Créez un nouveau projet.

- Ajoutez animal.json et équipement.json au projet.
- Créez le module `Modèle.py` ainsi qu'un fichier de test

Pensez également à conserver une copie de ces fichiers dans un répertoire `orig`, pour que si les mises à jour ratent, vous puissiez repartir de zéro.

A faire

1. Lecture

1.1 État

Créez dans `Modèle.py` une fonction `lit_état` qui:

- prend en paramètre le nom de l'animal
- consulte le fichier `animal.json`
- et retourne l'état de l'animal

```
import json

def lit_état(animal_id):
    . . .
```

- Si l'animal est dans la liste, la fonction retourne la valeur de son `"ETAT"`;
- Si l'animal n'est pas dans la liste, la fonction:
  - affiche un message du type `Désolé, XXX n'est pas un animal connu` (où XXX doit être remplacé par le nom de l'animal)
  - retourne `None` (état nul)

- Les informations sont situées dans un fichier au format `json`. Pour lire le contenu du fichier `animal.json`, il suffit d'ouvrir le fichier avec la commande suivante :

```
with open('animal.json', "r", encoding='utf-8') as f:
    animal = json.load(f)
```

on récupère ainsi un dictionnaire `animal` indexé par les identifiants des différents animaux (ici `Tic`, `Tac`, `Patrick`, `Totoro` et `Pocahontas`).

- Pour lire l'enregistrement complet d'un animal, on utilise l'adressage par identifiant :

```
poca = animal['Pocahontas']
```

- remarque : `poca` est lui-même un dictionnaire. Pour lire une valeur spécifique, on utilise l'adressage par attribut

```
poca_race = poca['RACE']
```

- ou de façon plus synthétique, la double indexation:

```
poca_race = animal['Pocahontas']['RACE']
```

La fonction `lit_état` doit passer les tests suivants:

```
import Modèle

def test_lit_etat():
    assert Modèle.lit_état('Tac') == 'affamé'

def test_lit_etat_nul():
    assert Modèle.lit_état('Bob') == None
```

- Le premier test vérifie que l'état (initial) de l'animal `Tac` est `"affamé"`.
- Le second test vérifie que l'état nul (`None`) est retourné lorsque le nom n'est pas dans le tableau

1.2 Lieu

Créez dans le Modèle une fonction `lit_lieu` analogue à `lit_état` qui retourne le lieu où se trouve l'animal.

```
def lit_lieu(animal_id):
    . . .
```

De manière analogue fonction devra passer les tests suivants:

```
def test_lit_lieu():
    assert Modèle.lit_lieu('Tac') == 'litière'

def test_lit_lieu_nul():
    assert Modèle.lit_lieu('Bob') == None
```

1.3 Disponibilité

Créer une fonction `vérifie_disponibilité` retourne la disponibilité du lieu indiqué à partir de la table des équipements.

```
def vérifie_disponibilité(équipement_id):
    . . .
```

Cette fonction devra passer les tests suivants :

```
def test_vérifie_disponibilité():
    assert Modèle.vérifie_disponibilité('litière') == 'libre'
    assert Modèle.vérifie_disponibilité('nid') == 'occupé'
```

- Si l'équipement n'est pas dans la liste, la fonction:
  - affiche un message du type `Désolé, XXX n'est pas un équipement connu` (où XXX doit être remplacé par le nom de l'équipement)
  - retourne `None` (état nul)

```
def test_vérifie_disponibilité_nul():
    assert Modèle.vérifie_disponibilité('nintendo') == None
```

1.4 Cherche_occupant

La fonction `cherche_occupant` retourne la liste des occupants du lieu indiqué à partir de la table des animaux.

```
def cherche_occupant(lieu):
    . . .
```

Elle devra passer les tests suivants:

```
def test_cherche_occupant():
    assert Modèle.cherche_occupant('nid') == ['Pocahontas']
    assert 'Tac' in Modèle.cherche_occupant('litière')
    assert 'Tac' not in Modèle.cherche_occupant('mangeoire')
```

- Si le lieu n'est pas référencé, la fonction:
  - affiche un message du type `Désolé, XXX n'est pas un lieu connu` (où XXX doit être remplacé par le nom du lieu)
  - retourne `None` (état nul)

```
def test_cherche_occupant_nul():
    assert Modèle.cherche_occupant('casino') == None
```

2. Ecriture

2.1 Changement d'état

Créez une fonction d'écriture `change_état` qui modifie l'état de l'animal à partir de son nom et d'un nouvel état.

```
def change_état(id_animal, état):
    . . .
```

La fonction devra passer les tests suivants :

```
def test_change_état():
    Modèle.change_état('Totoro', 'fatigué')
    assert Modèle.lit_état('Totoro') == 'fatigué'
    Modèle.change_état('Totoro', 'excité comme un pou')
    assert Modèle.lit_état('Totoro') == 'fatigué'
    Modèle.change_état('Bob', 'fatigué')
    assert Modèle.lit_état('Bob') == None
```

Autrement dit seuls les états `affamé`, `fatigué`, `repus`, `endormi` sont autorisés.

- Pour modifier de manière effective les changements effectués, il est bien sûr nécessaire de mettre à jour le fichier avec la fonction `json.dump`.

```
with open('animal.json', "w") as g:
    json.dump(animal, g)
```

- Pour sauver un fichier `json` plus joli, on peut utiliser le paramètre indent. Par exemple :

```
json.dump(animal, open("animal.json", "w"), indent=4)
```

2.2 Changement de lieu

Créez une fonction `change_lieu` permettant à un animal de changer de lieu.

```
def change_lieu(id_animal, lieu):
    . . .
```

Attention, à l'exception de la litière, chaque lieu ne peut être occupé que par un seul animal à la fois.

- Lorsqu'un animal quitte un lieu, celui-ci devient `libre`;
- Lorsqu'un animal entre dans un nouveau lieu, celui-ci devient `occupé` (à l'exception de la litière qui est toujours `libre`);
- Il faut bien sûr modifier l'attribut `LIEU` de l'animal.

La fonction `change_lieu` modifie à la fois le fichier `animal.json` **et** le fichier `équipement.json`.

La fonction devra passer les tests suivants:

```
def test_change_lieu():
    Modèle.change_lieu('Totoro', 'roue')
    assert Modèle.lit_lieu('Totoro') == 'roue'
    assert Modèle.vérifie_disponibilité('litière') == 'libre'
    assert Modèle.vérifie_disponibilité('roue') == 'occupé'
```

Si le lieu est occupé, la fonction

- ne change rien
- affiche le message `Désolé, le lieu XXX est déjà occupé`.

```
def test_change_lieu_occupé():
    Modèle.change_lieu('Totoro', 'nid')
```

```
    assert Modèle.lit_lieu('Totoro') == 'roue'
```

Si le lieu n'existe pas, la fonction ne change rien:

```
def test_change_lieu_nul_1():
    Modèle.change_lieu('Totoro', 'casino')
    assert Modèle.lit_lieu('Totoro') == 'roue'
```

Si l'animal n'existe pas, la fonction ne change rien non plus:

```
def test_change_lieu_nul_2():
    Modèle.change_lieu('Bob', 'litière')
    assert Modèle.lit_lieu('Bob') == None
```

2. Le Contrôleur

Le contrôleur est la partie du programme qui met en œuvre l'ensemble des actions possibles:

- `nourrir`
- `divertir`
- `coucher`
- `réveiller`

En vérifiant certaines contraintes d'intégrité:

- seul un animal affamé accepte d'être nourri
- seul un animal repus accepte de faire de l'exercice
- seul un animal fatigué accepte de dormir
- seul un animal endormi peut être réveillé

Ajoutez à votre projet un nouveau module `Contrôleur.py` qui implémentera les fonctions `nourrir`, `divertir`, `coucher` et `réveiller`. Le contrôleur fait appel au module `Modèle` pour toutes les opérations de lecture et d'écriture.

A faire

1. Nourrir

Créez une fonction `nourrir` qui prend en argument un identifiant d'animal.

- Si la mangeoire est occupée, la fonction affiche un message du type `Impossible, la mangeoire est actuellement occupée par XXX`.
- Si l'animal n'est pas `affamé`, la fonction affiche un message du type `Désolé, XXX n'a pas faim!`.
- Si l'animal est affamé et la mangeoire est libre, alors:
  - L'animal est déplacé vers la `mangeoire`
  - L'animal devient `repus`
  - L'état de la mangeoire devient `occupé`

- (Sinon rien ne change)

La fonction `nourrir` devra passer les tests suivants :

```
import Contrôleur

def test_nourrir():
    if Modèle.vérifie_disponibilité('mangeoire') == 'libre' and
Modèle.lit_état('Tic') == 'affamé':
        Contrôleur.nourrir('Tic')
    assert Modèle.vérifie_disponibilité('mangeoire') == 'occupé'
    assert Modèle.lit_état('Tic') == 'repus'
    assert Modèle.lit_lieu('Tic') == 'mangeoire'
    Contrôleur.nourrir('Tac')
    assert Modèle.lit_état('Tac') == 'affamé'
    assert Modèle.lit_lieu('Tac') == 'litière'
    Contrôleur.nourrir('Pocahontas')
    assert Modèle.lit_état('Pocahontas') == 'endormi'
    assert Modèle.lit_lieu('Pocahontas') == 'nid'
    Contrôleur.nourrir('Bob')
    assert Modèle.lit_état('Bob') == None
    assert Modèle.lit_lieu('Bob') == None
    assert Modèle.vérifie_disponibilité('mangeoire') == 'occupé'
```

- A l'issue des tests, les trois messages suivants doivent s'afficher :
    - `Désolé, la mangeoire est occupée par ['Tic']`
    - `Désolé, Pocahontas n'a pas faim`
    - `Désolé, Bob n'est pas un animal connu`

2. Divertir

Créez et testez une fonction `divertir` qui prend en argument un identifiant d'animal.

- Si la roue est occupée, la fonction affiche un message du type `Impossible, la roue est actuellement occupée par XXX`.
- Si l'animal n'est pas `repus`, la fonction affiche un message du type `Désolé, XXX n'est pas en état de faire du sport!`.
- Si l'animal est repus et la roue est libre, alors:
    - L'animal est déplacé vers la `roue`
    - L'animal devient `fatigué`
    - L'état du lieu initial devient `libre`
- (Sinon rien ne change)

3. Coucher

Créez et testez une fonction `coucher` qui prend en argument un identifiant d'animal.

- Si le nid est occupé, la fonction affiche un message du type `Impossible, le nid est actuellement occupé par XXX`.

- Si l'animal n'est pas `fatigué`, la fonction affiche un message du type `Désolé, XXX n'est pas fatigué!`.
- Si l'animal est fatigué et le nid est libre, alors:
  - L'animal est déplacé vers le `nid`
  - L'animal devient `endormi`
  - L'état du lieu initial devient `libre`
- (Sinon rien ne change)

4. Réveiller

Créez et testez une fonction `réveiller` qui prend en argument un identifiant d'animal.

- Si l'animal n'est pas `endormi`, la fonction affiche un message du type `Désolé, XXX ne dort pas!`.
- Si l'animal est endormi, alors:
  - L'animal est déplacé vers la `litière`
  - L'animal devient `affamé`
  - L'état du nid devient `libre`
- (Sinon rien ne change)

3. La Vue

Cette partie sera vue dans un prochain TD!