Agilité : programmation par contrat

TP6 Agilité

Contract programming

Éric Ramat & Jérôme Buisine
eric.ramat@univ-littoral.fr
jerome.buisine@univ-littoral.fr

14 novembre 2022

Durée : 3 heures

L'objectif de ce TP est de proposer de nouveaux développements au sein du projet en appliquant de la programmation par contrat.

1 Travail

Ce support propose d'intégrer de vérifier le bon comportement d'un simulateur de parachutistes. De ce fait, vous allez dans ce TP développer différentes étapes en respectant des consignes spécifiques :

— 1. À partir d'une branche **`feature/ContractTaskX`** spécifique à une tâche, écrire pour chaque fonction ou méthode le contrat qui permet sa vérification.

— 2. Écrire ensuite le code associé.

— 3. Pour chaque code développé sous contrat, vérifiez que celui-ci fonctionne localement.

— 4. Ajouter un test unitaire qui permet de le vérifier sur des données aléatoires.

— 5. Soumettre en ligne la modification apportée et vérifier que le pipeline d'intégration continue fonctionne correctement.

Pour vous aider : voici un ensemble de ressources qui peuvent vous être utiles pour ce TP :

— 1. Documentation officielle de l'outil `Git`.

— 2. Gitlab : interface web pour la gestion de projets versionnés sous `Git`.

— 3. deal : une bibliothèque Python pour la conception par contrat (DbC) et la vérification des valeurs.

— 4. typings : documentation pour l'utilisation du typages (*Python Hint*).

Remarque importante : faites attention au copier/coller à partir du PDF qui peut prendre en compte des caractères inattendus et causer des erreurs.

2 Configuration de l'environnement

Nous allons premièrement configurer l'ensemble du projet `paratrooper`.

2.1 Le projet paratrooper

Tout d'abord, depuis l'interface Gitlab il vous faudra **cloner** le projet suivant : paratrooper-etudiant (**ne réalisez pas un fork de celui-ci !**). Il s'agit du projet sur lequel vous allez travailler et contenant une structure de base.

Créer un projet **privé** Gitlab avec la convention de nommage suivante : « paratrooper-YOUR_NAME » avec `YOUR_NAME` composé de la première de votre prénom puis votre nom. Exemple, pour Jean Dupont, on obtient : `jdupont`, soit « paratrooper-jdupont ». Il faudra rajouter ensuite en tant que rapporteurs du projet : `jbuisine` et `eramat`.

Récupérez l'URL `git` de projet Gitlab créé et configurez l'URL `git` au sein du projet cloné depuis votre terminal :

```
git remote set-url origin "YOUR_URL.git"
```

Important : vous devez à ce stade configurer un environnement de travail propre de Python. Pour cela, référez-vous à la documentation suivante : environnement Python. Elle vous permet la mise en place d'un environnement virtuel Python. Nous utiliserons pour ce projet une version 3.8.0 de Python.

Vous pouvez vérifier que l'application fonctionne correctement en exécutant le programme principal :

```
python src/main.py
```

2.2 Première utilisation de deal

Pour intégrer des contrats, il vous faudra utiliser les dépendances `hypothesis` et `deal`. Vérifiez qu'elles sont bien présentes dans votre projet :

```
hypothesis==6.56.1
deal==4.23.4
```

Vous pouvez vérifier son fonctionnement à partir du code Python suivant (à exécuter à partir d'un script à part) :

```
import deal

@deal.ensure(lambda _: _.result.startswith(_.left))
@deal.ensure(lambda _: _.result.endswith(_.right))
@deal.ensure(lambda _: len(_.result) == len(_.left) + len(_.right))
@deal.has()
def concat(left: str, right: str) -> str:
        return left + right

concat("this is", "the first contract")
```

Important : pour la suite du TP, on considère que les contrats ne sont pas vérifiés sur l'environnement de production. Modifiez en conséquence la configuration de la CI/CD.

3 Développements

Pour rappel, la librairie `deal` ne permet pas de réaliser des preuves de contrat sur des méthodes d'instances. Seules les fonctions dites simples et méthodes statiques peuvent être prouvées. Il est toutefois possible d'utiliser l'instruction `deal.cases` pour réaliser des tests sur des jeux de données aléatoires.

Histoire 1 : On souhaite que les propriétés de force de vent soient inchangées et ne varient pas au cours de l'exécution du programme.

Indications :
- Il faudra cibler les propriétés de la classe `Wind`.

Histoire 2 : À la création d'un vent aléatoire, on souhaite s'assurer que la force maximale du vent ne peut pas être dépassée et que la direction du vent est telle qu'attendue.

Indications :
- Il faudra cibler la méthode `random` de la classe `Wind`.
- Il sera demandé pour ce test de le vérifier sur 100 jeux de données aléatoires.

Histoire 3 : Lors de la simulation physique de la chute d'un parachutiste, la coordonnée actuelle du parachutiste doit être différente de (0, 0). Le vent doit être positif. La coordonnée d'arrivée doit être différente de la coordonnée initiale.

Indications :
- Il faudra cibler la fonction `estimated_fall` du module `base.utils`.
- Il sera demandé pour ce test de le vérifier sur 100 jeux de données aléatoires.

Histoire 4 : Lors de l'ajout d'un parachutiste dans le monde, on souhaite vérifier que son vol est bien enregistré. Il faudra également s'assurer qu'il n'est pas associé à plusieurs vols.

Indications :
- Il faudra cibler la méthode `add_fly` de `World`.
- Il sera demandé pour ce test de le vérifier sur 50 jeux de données aléatoires.
- Vous pouvez utiliser les fonctions `sum` et `any` proposées par Python.

Histoire 5 : Lors de l'avancée de la simulation, on souhaite récupérer les parachutistes encore en vol ou arrivés.

Indications :
- Il faudra cibler la méthode `step` de `World`.
- Il sera demandé pour ce test de le vérifier sur 50 jeux de données aléatoires.
- Il faudra vérifier quel état du parachutiste n'est pas attendu.

Histoire 6 : Lorsque le parachutiste est sur le sol, on s'attend à ce que sa coordonnée en `y` soit de 0 (altitude).

Indications :
- Il faudra cibler la propriété `state` du parachutiste qui sera automatiquement vérifiée par les tests.

4 Remise des travaux

N'oubliez pas de réaliser un dernier commit de vos travaux. Taguez ensuite votre projet avec le tag « `tp6` » et soumettez-le sur le serveur Gitlab. Vérifiez que le pipeline reste fonctionnel. Cette version de projet fera office de rendu.