

Critères d'évaluation

1. Soumission d'un bundle valide avec historique complet (10%)

Vous avez soumis sur le serveur du cours un bundle Git **valide** qui contient l'historique de développement de votre projet. Cet historique doit comporter **au moins cinq** (5) révisions distinctes et significatives. Ceci signifie notamment que vous ne devez **pas** attendre la fin de votre projet avant de faire votre première révision («*commit*») dans votre dépôt. Cela implique aussi que vous fassiez **régulièrement** des révisions, c'est-à-dire à chaque fois que vous complétez une étape de votre développement.

Seules les révisions associées à votre adresse courriel ULaval seront considérées, voir la [FAQ](#) du forum de discussion pour des détails sur comment configurer votre dépôt Git.

La branche « *master* » de votre dépôt, c'est-à-dire la branche par défaut, contient bien les fichiers `main.py`, `gobblet.py` et `api.py`, tel que spécifié par l'énoncé du projet. De façon optionnelle, votre projet peut aussi contenir d'autres fichiers de type `*.py`, des fichiers `*.md` ou le fichier `.gitignore`. Aucun autre type de fichier n'est permis.

2. Respect des normes de programmation (15%)

Votre programme respecte **tous** les préceptes du [PEP-8](#).

L'évaluation de ce critère sera basée sur le rapport [pylint](#) produit lors de la soumission de votre projet (voir onglet «*Soumission*»). N'attendez pas le jour de la remise avant de vérifier la conformité de votre programme. Vous devez **minimiser** le nombre de messages produit par ce rapport. Votre note pour ce critère sera **proportionnelle** à la note produite par pylint, mais **pas** nécessairement linéairement.

Avertissement: si pour une raison quelconque votre code source fait **planter** pylint (ceci sera indiqué dans le rapport de soumission), ou si le temps d'exécution de pylint est **trop long**, ce qui est aussi symptomatique d'un problème avec votre code source, vous obtiendrez **zéro** pour ce critère. Le cas échéant, assurez-vous de **poser des questions** sur le forum afin de résoudre le problème bien **avant** la date limite de soumission. Notez aussi que s'il y a présence d'erreurs PEP-8 dans la coquille fournie, vous **devez** les corriger avant de soumettre votre projet.

3. Succès des tests unitaires (5%)

Votre solution passe avec succès tous les tests unitaires fournis dans la coquille du projet.

4. Interface ligne de commande (5%)

Votre solution analyse la ligne de commande en définissant dans votre module `gobblet` une fonction nommée `interpréteur_de_commande`, tel que demandé par l'énoncé du projet. Cette fonction retourne bien le résultat produit par la méthode `parse_args` de la classe `argparse.ArgumentParser`. L'objet retourné contient bien les attributs `lister` et `IDUL`, tel que spécifié par l'énoncé du projet.

Lorsque l'utilisateur spécifie l'option `-h`, la fonction affiche de l'aide selon le format demandé.

5. Affichage du jeu (20%)

Votre solution définit dans le module `gobblet` les fonctions nommées `formater_un_gobblet`, `formater_un_joueur`, `formater_plateau`, et `formater_jeu`, tel que spécifié par l'énoncé du projet.

Ces fonctions affichent bien le `plateau` d'une partie à partir de son `état`, en respectant **scrupuleusement** le format d'affichage spécifié.

6. Formattage des parties (10%)

Votre solution définit dans le module `gobblet` une fonction nommée `formater_les_parties`, tel que spécifié par l'énoncé du projet. Cette fonction produit bien la liste des parties reçue en argument, en respectant scrupuleusement le format d'affichage spécifié.

7. Récupération du coup (10%)

Votre solution définit dans le module `gobblet` une fonction nommée `récupérer_le_coup`, tel que spécifié par l'énoncé du projet.

Cette fonction ne pose que deux (2) questions et retourne bien l'origine et la destination tel que spécifié dans l'énoncé.

8. Communications avec le serveur (20%)

Votre module `api` contient bien les quatre fonctions suivantes:

1. `lister_parties`
2. `débuter_partie`
3. `récupérer_partie`
4. `jouer_coup`

qui permettent d'établir la **communication** avec le serveur de jeu, tel que décrit par l'énoncé du projet.

9. Qualité du code (5%)

Le code source de votre solution est de bonne qualité selon diverses métriques de performance et de

Jeu Gobblet - Phase 1

Dans le cadre des projets de cette session, nous allons développer un programme pour jouer au jeu Gobblet. Dans un premier temps, vous devez vous familiariser avec les [règles de ce jeu](#) qui utilise un plateau composé d'une grille 4×4 . L'image suivante illustre le plateau en question :



Règles du jeu

Principe et but du jeu

Le plateau de jeu est donc composé d'une grille de 4×4 cases et chaque joueur dispose initialement d'un inventaire de trois (3) piles de quatre (4) gobelets emboîtés les uns dans les autres.

Déroulement d'une partie

À son tour, chaque joueur doit déplacer un gobelet pour le positionner sur le plateau. Le gobelet peut provenir soit:

- d'une des trois piles du joueur;
- d'un gobelet déjà positionné sur le plateau.

Le gobelet doit ensuite être placé sur le plateau de jeu, soit:

- à un emplacement libre;
- à un emplacement déjà occupé par un gobelet de taille inférieure.

Fin de la partie

Le premier joueur qui parvient à produire une ligne de 4 gobelets lui appartenant, que ce soit une ligne horizontale, verticale ou diagonale, gagne la partie.

Détails importants

- L'**origine** d'un gobelet peut être la **pile** du joueur ou le **plateau**.
- La **destination** d'un gobelet est **obligatoirement** le plateau.
- Seul un gobelet de taille supérieur peut être placé par-dessus un gobelet de taille inférieure.
- Seul le gobelet visible peut être déplacé, que ce soit depuis les piles du joueur ou depuis le plateau.
- Pour plus de détail, voir cette [vidéo explicative](#) et lire les [règles officielles](#) du jeu.

A faire

Pour cette première phase de développement, **avant** la date limite ci-dessus (haut de la page), vous devez accomplir les tâches suivantes :

1. **Installer** sur votre ordinateur l'éditeur de code nommé [VS Code](#). Il s'agit d'un environnement de développement moderne, efficace et convivial. Vous trouverez de plus amples informations sur l'installation et la configuration de ce logiciel dans la [FAQ](#) du forum de discussion.
2. **Installer** sur votre ordinateur le logiciel nommé [Git](#) pour la gestion des révisions de votre projet. De nouveau, vous trouverez de plus amples informations sur l'usage de ce logiciel dans la [FAQ](#) du forum. Notez que l'usage de Git est **obligatoire** pour que vous puissiez soumettre votre projet sur le site du cours.
3. **Créer** votre équipe de travail. Pour cette phase 1, vous devez **obligatoirement** faire le travail en solo, mais vous devez malgré tout procéder à la formation d'une équipe. Pour ce faire, allez simplement à l'onglet *Équipe*, entrez un pseudonyme pour votre équipe et cliquez sur le bouton de création. À partir de là, vous pourrez soumettre le fruit de votre travail à l'onglet *Soumissions*.
4. **Écrire** onze (11) fonctions spécifiques dans deux (2) modules distincts. Ces fonctions sont toutes relativement courtes et simples, une fois que l'on a compris ce qui est demandé. Ne cherchez pas midi à quatorze heures, et n'hésitez pas à poser des questions dans le forum. Voir ci-dessous pour le détail de ces fonctions. Assurez-vous de respecter **tous** les critères d'évaluation du projet (voir dans le haut de cette page).
5. **Produire** une archive **Git** (« *bundle* ») des différentes révisions de vos modules.

Module `main`

Tout d'abord, notez que votre programme principal doit **obligatoirement** se trouver dans un fichier nommé `main.py` et que celui-ci doit se trouver à la racine de votre dépôt Git, dans la branche « *master* » (la branche par défaut).

Votre programme principal doit faire en sorte que l'on puisse entrer l'idul du joueur sur la **ligne de commande**, de la façon suivante :

```
python main.py idul_du_joueur
```

où `idul_du_joueur` est votre **idul**. Pour ce faire, à l'intérieur du module `gobblet`, vous devez définir une fonction nommée `interpréteur_de_commande` (fonction 1 de 11) qui fera appel au module `argparse` (voir document afférent) afin d'interpréter la commande entrée depuis votre terminal. Votre fonction doit retourner l'objet produit par la fonction `parse_args` de la classe `argparse.ArgumentParser`, et cet objet doit contenir l'idul du joueur associé à la clé `IDUL` ainsi que la clé `lister`. Autrement dit, votre fonction doit avoir la forme :

```
▶ 1 import argparse
  2
  3
  4 def interpréteur_de_commande():
  5     """Interpréteur de commande
  6
  7     Returns:
  8         Namespace: Un objet Namespace tel que retourné par parser.parse_args().
  9             Cette objet aura l'attribut IDUL représentant l'idul du joueur
 10                et l'attribut lister qui est un booléen True/False.
 11
 12     """
 13     parser = ArgumentParser()
 14
 15     # Complétez le code ici
 16     # vous pourriez aussi avoir à ajouter des arguments dans ArgumentParser(...)
 17
 18     return parser.parse_args()
```

Et doit faire en sorte que l'option `--help` affiche le message suivant:

Serveur de jeu et module `api`

Accessible sur Internet à l'adresse <https://pax.ulaval.ca/>, le serveur du cours implante le jeu Gobblet. Votre programme doit interagir avec ce serveur afin de pouvoir jouer contre un adversaire humain ou robotisé. Pour ce faire, le serveur possède ce qu'on appelle en anglais un [web API](#). Cette interface web possède les quatre ressources suivantes :

1. `GET gobblet/api/parties/`: pour lister les identifiants de vos parties précédentes ;
2. `GET gobblet/api/partie/<id_partie>`: pour récupérer l'état d'une partie spécifique à partir de son identifiant ;
3. `POST gobblet/api/partie/`: pour débuter une nouvelle partie ;
4. `PUT gobblet/api/jouer/`: pour jouer un coup dans une partie en cours.

Pour communiquer avec le serveur de jeu, vous allez installer et utiliser le module Python nommé `requests` qui permet de facilement faire des requêtes à un serveur en utilisant le protocole [HTTP](#). Par exemple, les énoncés suivants permettent de faire une requête `GET` sur le serveur web de l'Université Laval, afin d'obtenir le contenu en [HTML](#) de sa page d'accueil:

```
1 import requests
2
3 URL = 'http://www.ulaval.ca/'
4
5 # récupérer le contenu de la ressource
6 rep = requests.get(URL)
7
8 if rep.status_code == 200:
9     # la réponse est bonne, afficher son contenu
10    print(rep.text)
11
12 else:
13    # une erreur s'est produite, afficher le message d'erreur
14    print(f"Le GET sur {URL} a produit le code d'erreur {rep.status_code}.")
```

Comme vous pouvez le constater, cette page contient du code HTML plus ou moins compréhensible pour un humain, mais que votre fureteur Web est capable de facilement afficher.

La fonction `get` du module `requests` retourne donc un objet qui contient différentes informations dont notamment un `code de statut` ainsi que le `texte de la réponse`. Le `code HTTP` 200 signifie que la requête s'est déroulée sans erreur. Dans ce cas, le texte de la réponse se trouve associé à l'attribut `text` de l'objet retourné par la requête.

Nous décrivons maintenant les quatre (4) **requêtes** spécifiques que vous devez supporter en définissant quatre (4) **fonctions** correspondantes dans votre module `api` (fichier nommé `api.py`). Ces fonctions doivent être nommées :

1. `lister_parties`
2. `débuter_partie`
3. `récupérer_partie`
4. `jouer_coup`

L'authentification

Pour communiquer avec le serveur de jeu, vous aurez besoin de lui passer vos informations d'authentification. Ces informations sont :

- Votre `idul`;
- Votre jeton personnel.

Ces informations seront passées sous forme de tuple `(idul, secret)` au paramètre `auth` lors de l'appel au serveur comme démontré plus loin.

Le jeton personnel n'est pas votre mot de passe, mais bien le jeton que vous pouvez copier depuis votre tableau de bord sur le site du cours. Vous pouvez, par exemple, l'assigner à la variable `SECRET` du module `main` pour l'utiliser dans votre boucle de jeu.

`gobblet/api/parties/` (type GET)

Cette requête de type `GET` s'attend en entrée à recevoir deux (2) paramètres `idul` et `secret` comme authentification, ces paramètres identifient l'`idul` du joueur et le jeton personnel de ce dernier. En cas de succès (code `200`), le serveur retourne en JSON un **dictionnaire** contenant la clé suivante :

1. `parties`: une liste des (max) 20 parties les plus récentes de l'usager ;

où chaque partie dans la liste est elle-même un dictionnaire contenant les clés suivantes :

1. `id`: l'identifiant de la partie ;
2. `date`: la date de création de la partie ;
3. `joueurs`: la liste ordonnée des joueurs ;
4. `gagnant`: le nom du gagnant si existant, sinon `None`.

En cas d'erreur, si le code de votre réponse est `401` ou `406`, elle retourne en JSON un **dictionnaire** contenant la clé suivante :

1. `message`: un message qui décrit la nature de l'erreur en question ;

Par exemple, pour un `idul` `xxxxx` et un `secret` `"80a0a0d2-059d-4539-9d53-78b3f6045943"` :

```
> 1 import requests
  2
  3 URL = 'https://pax.ulaval.ca/gobblet/api/'
  4
  5 rep = requests.get(URL+'parties', auth=("xxxxx", "80a0a0d2-059d-4539-9d53-78b3f6045943"))
  6
  7 if rep.status_code == 200:
  8     # La requête s'est déroulée normalement;
  9     # décoder le JSON et afficher la liste de parties
 10    rep = rep.json()
 11    print(rep)
 12
 13 elif rep.status_code == 401:
 14     # Votre requête est invalide;
 15     # décoder le JSON et afficher le message d'erreur
 16    rep = rep.json()
 17    print(rep)
 18
 19 else:
 20     # Une erreur inattendue est survenue
 21     print(f"Le GET sur '{URL}parties' a produit le code d'erreur {rep.status_code}.")
```

Dans votre module Python `api`, vous devez définir une fonction nommée `lister_parties` (fonction 8 de 11) qui accepte en entrée un `Idul` et un `secret`, et qui retourne en sortie la liste des parties reçues du serveur, après avoir décodé le JSON de sa réponse. En cas d'erreur (code `401`), votre fonction doit soulever une exception de type `PermissionError` avec le message reçu. En cas d'erreur (code `406`), votre fonction doit soulever une exception de type `RuntimeError` avec le message reçu. En cas de code de statut autre que `200`, `401` ou `406`, votre fonction doit soulever une exception de type `ConnectionError`, vous n'avez pas à mettre de message pour ce type d'exception.

`gobblet/api/partie/<id_partie>` (type GET)

Cette requête de type `GET` s'attend en entrée à recevoir un l'identifiant `id_partie` de la partie à même l'url, ainsi qu'un `idul` et un `secret` pour l'authentification. En cas de succès (code `200`), le serveur retourne en JSON un **dictionnaire** contenant les clés suivantes :

1. `id`: l'identifiant de la partie en cours ;
2. `plateau`: l'état du plateau ;
3. `joueurs`: l'état des joueurs ;
4. `gagnant`: le nom du joueur gagnant, `None` s'il n'y a pas encore de gagnant.

En cas d'erreur (code `401` ou `406`), le serveur retourne en JSON un **dictionnaire** contenant la clé suivante :

1. `message`: un message qui décrit la nature de l'erreur en question.

Exemple d'url pour la partie `80a0a0d2-059d-4539-9d53-78b3f6045943` :

```
"https://pax.ulaval.ca/gobblet/api/partie/80a0a0d2-059d-4539-9d53-78b3f6045943"
```

Dans votre module Python `api`, vous devez définir une **fonction** nommée `récupérer_partie` (fonction 9 de 11) qui **accepte** en entrée un `id_partie` sous la forme d'une chaîne de caractères ainsi que l'`idul` et le `secret`, et qui **retourne** en sortie un **tuple** constitué de l'identifiant de la partie, de l'état du plateau et de l'état des joueurs. En cas d'erreur `401`, votre fonction doit **soulever** une exception de type `PermissionError` avec le message reçu. En cas d'erreur `406`, votre fonction doit **soulever** une exception de type `RuntimeError` avec le message reçu. En cas de code de statut autre que `200`, `401` ou `406`, votre fonction doit **soulever** une exception de type `ConnectionError`, vous n'avez pas à mettre de message pour ce type d'exception.

`gobblet/api/partie/` (type POST)

Cette requête est de type `POST`, contrairement aux deux requêtes précédentes, car elle modifie l'état interne du serveur en créant une nouvelle partie. Elle s'attend à recevoir les mêmes éléments d'authentification que précédemment. En cas de succès (code `200`), le serveur retourne en JSON un **dictionnaire** contenant les clés suivantes :

gobblet/api/partie/ (type POST)

Cette requête est de type **POST**, contrairement aux deux requêtes précédentes, car elle modifie l'état interne du serveur en créant une nouvelle partie. Elle s'attend à recevoir les mêmes éléments d'authentification que précédemment. En cas de succès (code **200**), le serveur **retourne** en JSON un **dictionnaire** contenant les clés suivantes :

1. **id**: l'identifiant de la partie en cours ;
2. **plateau**: l'état du plateau ;
3. **joueurs**: l'état des joueurs ;
4. **gagnant**: le nom du joueur gagnant, **None** s'il n'y a pas encore de gagnant.

En cas d'erreur (code **401** ou **406**), le serveur **retourne** en JSON un **dictionnaire** contenant la clé suivante :

1. **message**: un message qui décrit la nature de l'erreur en question.

Dans votre module Python **api**, vous devez définir une **fonction** nommée **débuter_partie** (fonction 10 de 11) qui **accepte** en entrée un **idUser** et un **secret** et qui **retourne** en sortie un **tuple** constitué de l'identifiant de la partie, de l'état du plateau et de l'état des joueurs. En cas d'erreur **401**, votre fonction doit **soulever** une exception de type **PermissionError** avec le message reçu. En cas d'erreur **406**, votre fonction doit **soulever** une exception de type **RuntimeError** avec le message reçu. En cas de code de statut autre que **200**, **401** ou **406**, votre fonction doit **soulever** une exception de type **ConnectionError**, vous n'avez pas à mettre de message pour ce type d'exception.

gobblet/api/jouer/ (type PUT)

En plus des informations d'authentification habituelles, cette requête de type **PUT** s'attend à recevoir en entrée les trois (3) informations suivantes :

1. **id**: l'identifiant de la partie en cours ;
2. **destination**: la position sur le plateau tel que **[1, 3]** ;
3. **origine**: le numéro de la pile tel que **0** ou la position sur le plateau tel que **[1, 3]**.

Et retourne en JSON un **dictionnaire** contenant les clés suivantes :

1. **id**: l'identifiant de la partie en cours ;
2. **plateau**: l'état du plateau ;
3. **joueurs**: l'état des joueurs ;
4. **gagnant**: le nom du joueur gagnant, **None** s'il n'y a pas encore de gagnant.

En cas d'erreur (code **401** ou **406**), elle retourne en JSON un **dictionnaire** contenant la clé suivante :

1. **message**: un message qui décrit la nature de l'erreur en question.

Pour transmettre une requête de type **PUT** avec le module **request**, il s'agit simplement d'appeler sa fonction **put** au lieu de **get** ou **post**, en utilisant l'argument nommé **json** pour lui passer les informations. Par exemple :

```
1 rep = requests.put(URL+'partie', auth=("xxxxx", "80a0a0d2-059d-4539-9d53-78b3f6045943") json={"id": "11a0a0d2-059d-4539-9d53-78b3f6045943"}  
2 rep.json()
```

Lorsqu'un code **416** est retourné par la réponse, c'est que l'association **idul/secret** n'est pas valide. Lorsqu'un code **406** est retourné par la réponse, c'est que le coup spécifié par le joueur n'est pas valide et n'a pas été joué et l'état du jeu restera **Inchangé** sur le serveur. Si le code est **200**, le coup du joueur a été joué et l'état de la partie incorpore la réponse du robot. C'est alors encore au tour du joueur de choisir son prochain coup. Lorsqu'un gagnant est déclaré, la partie est terminée et l'état du jeu incorpore le **coup gagnant**.

Lorsqu'un code `416` est retourné par la réponse, c'est que l'association `idul/secret` n'est pas valide. Lorsqu'un code `406` est retourné par la réponse, c'est que le coup spécifié par le joueur n'est pas valide et n'a pas été joué et l'état du jeu restera inchangé sur le serveur. Si le code est `200`, le coup du joueur a été joué et l'état de la partie incorpore la réponse du robot. C'est alors encore au tour du joueur de choisir son prochain coup. Lorsqu'un gagnant est déclaré, la partie est terminée et l'état du jeu incorpore le coup gagnant.

Dans votre module Python `api`, vous devez définir une fonction nommée `jouer_coup` (fonction 11 de 11) qui accepte en entrée les cinq (5) arguments `id_partie`, `origine`, `destination`, `idul` et `secret`, et qui retourne en sortie un tuple constitué de l'identifiant de la partie, de l'état du plateau et de l'état des joueurs. En cas d'erreur `401`, votre fonction doit soulever une exception de type `PermissionError` avec le message reçu. En cas d'erreur `406`, votre fonction doit soulever une exception de type `RuntimeError` avec le message reçu. En cas de code de statut autre que `200`, `401` ou `406`, votre fonction doit soulever une exception de type `ConnectionError`, vous n'avez pas à mettre de message pour ce type d'exception. Dans le cas d'une partie terminée (présence d'un gagnant dans la réponse), elle doit soulever une exception de type `StopIteration` avec le nom du gagnant comme message de l'exception.

```
usage: main.py [-h] [-l] IDUL

Gobblet

positional arguments:
  IDUL      IDUL du joueur

optional arguments:
  -h, --help    show this help message and exit
  -l, --lister  Lister les parties existantes
```

Voir le document qui effectue une introduction au [module argparse](#) pour de plus amples détails.

Par ailleurs, votre programme principal doit :

1. contacter le serveur afin de débuter une partie et recevoir l'état initial du jeu ;
2. afficher à la console l'état actuel du jeu sous la forme d'un damier en « [art ascii](#) » ;
3. demander à l'utilisateur de spécifier son coup ;
4. transmettre le coup en question au serveur et recevoir le nouvel état du jeu ;
5. boucler sur l'étape 2.

Notez que dans la coquille Github que nous vous fournissons afin de bien démarrer votre projet (voir à la fin de l'énoncé), nous avons inclus une boucle de jeu que vous pouvez modifier à votre guise. Notez aussi que même si le fichier [main.py](#) ne sera pas directement corrigé lors de la phase 1 du projet, tout code s'y trouvant doit respecter le PEP-8.

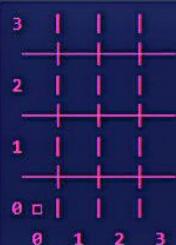
L'état du jeu vous sera transmis par le serveur sous la forme d'un **dictionnaire** de deux (2) éléments codé en [JSON](#) où le premier élément représente le plateau et le deuxième élément les joueurs. Notez que toutes les listes sont ordonnées. Voici un exemple de ce dictionnaire :

```
{  
    'plateau': [  
        [], [], [], []],  
        [], [], [], []],  
        [], [], [], []],  
        [], [], [], []],  
        [], [], [], []]  
    ],  
    'joueurs': [  
        {  
            'nom': 'jowick42',  
            'piles': [[1, 3], [1, 3], [1, 3]]}  
        },  
        {  
            'nom': 'robot',  
            'piles': [[2, 3], [2, 3], [2, 3]]}  
        }  
    ]  
}
```

La clé "`plateau`" correspond à l'état visible actuel du plateau de jeu. Il s'agit d'une liste de lignes, chacune formée d'une liste de cases, ces cases étant elles-mêmes formées d'une liste des gobelets actuellement présents sur ces cases. Par convention, la case située en bas à gauche du plateau correspond à la position (0, 0), soit l'élément `plateau[3][0]` dans la liste de listes. Une case vide est représentée par une liste vide, et une case avec un gobelet est représentée par une liste de deux valeurs: le numéro du joueur (1 ou 2) et la taille de son gobelet visible (0, 1, 2 ou 3):

```
[  
  [], [], [], []],  
  [], [], [], []],  
  [], [], [], []],  
  [[1, 3], [], [], []]  
]
```

Dans l'exemple ci-dessus, nous avons donc un plateau comportant un gobelet de taille 3 appartenant au joueur 1 et positionné en (0, 0) :



La clé "joueurs" correspond quant à elle à une liste ordonnée des joueurs, où le premier élément de la liste correspond au joueur 1 et le 2e au joueur 2. La clé "nom" correspond au nom du joueur (IDUL ou nom d'un robot). La clé "piles" correspond à une liste des trois (3) piles du joueur en question, le premier élément correspond à la pile 0 et le dernier à la pile 2. Une pile est soit une liste vide, donc pas de gobelet, ou une liste composée de deux (2) entiers : le numéro du joueur suivi de la taille du gobelet actuellement sur le dessus de pile. Vous avez accès uniquement au gobelet visible sur le dessus de la pile :

```
{  
  'nom': 'jowick42',  
  'piles': [[1, 1], [], [1, 3]]  
}
```

ce qui représente donc :

```
0 1 2  
jowick42: ◇ ◇
```

Module gobblet

En plus de la fonction `interpréteur_de_commande` décrite précédemment et devant se trouver dans le module `main`, votre module `gobblet` doit définir six (6) fonctions permettant de gérer l'affichage d'un plateau de jeu.

Une fonction nommée `formater_un_gobblet` (fonction 2 de 11) qui reçoit une liste vide ou une liste de deux (2) entiers [x, y] représentant un gobelet. Cette fonction doit retourner une représentation du gobelet pour le bon joueur. Les représentations sont disponibles dans la constante `GOBBLET_REPRÉSENTATION` du même fichier :

```
GOBBLET_REPRÉSENTATION = {  
    1: ["¤", "◊", "○", "¤"],  
    2: ["*", "◆", "●", "■"],  
}
```

où la clé 1 représente le joueur 1, la clé 2 le joueur 2 et chaque caractère de la liste représente un gobelet différent de taille 0 à 3 respectivement.

Par exemple:

```
>>> print(formater_un_gobblet([1, 2]))  
○  
>>> print(formater_un_gobblet([]))
```

Une fonction nommée `formater_un_joueur` (fonction 3 de 11) qui reçoit un dictionnaire représentant le joueur. Cette fonction doit retourner une représentation du joueur et doit obligatoirement faire appel à la fonction `formater_un_gobblet` pour formater les gobelets de la pile :

```
>>> print(formater_plateau([[[], [], [], []], [[], [], [], []], [[], [], [], []], [[1, 3], [], [], []]]))
```

3			
2			
1			
0	□		
0	1	2	3

Une fonction nommée `formater_jeu` (fonction 5 de 11) qui reçoit une liste représentant le plateau ainsi que la liste des joueurs. Cette fonction doit retourner une représentation du jeu et doit **obligatoirement** faire appel à la fonction `formater_plateau` ainsi qu'à la fonction `formater_un_joueur` :

```
>>> print(formater_jeu(  
    [[[], [], [], []], [[], [], [], []], [[], [], [], []], [[], [], [], []]],  
    [{'nom': 'jowick42', 'piles': [[1, 3], [1, 3], [1, 3]]}, {'nom': 'robot', 'piles': [[2, 3], [2, 3], [2, 3]]}]  
)  
          0   1   2  
jowick42:  □   □   □  
        robot: ■ ■ ■  
  
3 | | |  
---+---+---  
2 | | |  
---+---+---  
1 | | |  
---+---+---  

```

Votre module `gobblet` doit définir une fonction nommée `formatter_les_parties` (fonction 6 de 11) qui accepte en argument une liste de parties sous forme de dictionnaires comme ci-dessous :

```
[  
  {  
    "id": "5559cafd-6966-4465-af6f-67a784016b41",  
    "date": "2021-01-23 11:58:20",  
    "joueurs": ["jowic42", "robot"],  
    "gagnant": None  
  },  
  ...  
  {  
    "id": "80a0a0d2-059d-4539-9d53-78b3f6045943",  
    "date": "2021-01-24 14:23:59",  
    "joueurs": ["jowic42", "robot"],  
    "gagnant": "jowic42"  
  }  
]
```

et qui **retourne** en sortie une représentation textuelle de cette liste comme ceci :

```
"1 : 2021-01-23 11:58:20, jowic42 vs robot-2\n...\\n20: 2021-01-24 14:23:59, jowic42 vs robot-1, gagnant: jowic42"
```

Lorsqu'affichée par `print`, cette représentation donnera le résultat suivant :

```
1 : 2021-01-23 11:58:20, jowic42 vs robot-2  
...  
20: 2021-01-24 14:23:59, jowic42 vs robot-1, gagnant: jowic42
```

Veuillez à respecter l'ordre des parties reçues du serveur de jeu.

Notez que les ... désignent la présence d'autres parties que nous n'affichons pas ici afin de ne pas alourdir l'énoncé; vous devez formater toutes les parties reçues.

La dernière fonction de ce module est la fonction `récupérer_le_coup` (fonction 7 de 11) qui, en utilisant la fonction `input`, pose deux (2) questions à l'utilisateur: l'origine du gobelet à déplacer et sa destination sur le tableau de jeu.

Un couple d'entiers représentant une position sera "0,1" alors qu'une position dans la pile "0".

Par exemple:

```
>>> origin, destination = récupérer_le_coup()
Donnez le numéro de la pile (p) ou la position sur le plateau (x,y): 0
Où voulez-vous placer votre gobelet (x,y): 0,1
>>> print(origine)
0
>>> print(destination)
[0, 1]
```

```
>>> origin, destination = récupérer_le_coup()
Donnez le numéro de la pile (p) ou la position sur le plateau (x,y): 2,3
Où voulez-vous placer votre gobelet (x,y): 0,1
>>> print(origine)
[2, 3]
>>> print(destination)
[0, 1]
```

Notez qu'un fichier `tests.py` vous est fourni et possède divers tests unitaires qui vous serviront à mieux comprendre ce qui vous est demandé pour les fonctions d'affichage.