Mastermind

Enoncé du projet individuel

`rev.00 - 19 novembre 2015`

Contenu

Résumé

Ce document décrit le projet que nous vous demandons de réaliser de manière individuelle. Il s'agit de la programmation d'une variante à un seul joueur du Mastermind. Le document contient également les consignes, les modalités de remise ainsi que les critères d'évaluation du projet.

Introduction

Le but du projet est d'implémenter en *Python 3* une version simplifiée, à un joueur, du Mastermind. [1]

1 Fonctionnement général du jeu

La description suivante fait référence à l'interface décrite en Section A. Il est impératif pour la validation de votre code par UpyLaB que cette interface soit respectée à la lettre près.

Le jeu se déroule de la manière suivante (cf. exemple de déroulement en Section B) :

— Le moteur de jeu génère au hasard une combinaison de 4 pions de couleurs **différentes**, choisies parmi les associations lettre-couleur suivantes :

`r`	→ *rouge*	`j`	→ *jaune*	`v`	→ *vert*	`b`	→ *bleu*
`o`	→ *orange*	`c`	→ *blanc*	`t`	→ *violet*	`f`	→ *fuchsia*

— Le joueur a 10 essais pour deviner la combinaison. Pour chaque essai :

  — Le joueur introduit une combinaison (Q1, voir Section A).

  — L'ordinateur évalue cette combinaison et indique : (1) le nombre de pions de la bonne couleur bien placés et (2) le nombre de pions de la bonne couleur mais mal placés (M3). L'ordinateur peut, par exemple, indiquer que deux pions sont de la bonne couleur et bien placés, un pion est de la bonne couleur et mal placé, et le joueur en déduira qu'un pion est erroné. [2]

1. Voir `http://fr.wikipedia.org/wiki/Mastermind` pour les règles complètes.
2. Si, par exemple, un pion rouge est dans la combinaison à deviner et que le joueur propose plus d'un pion rouge, on considère qu'un seul pion rouge à la bonne couleur ou éventuellement est bien placé.

— Si, au lieu d'une combinaison, le joueur tape "`Q`" (pour *quitter*), la partie est interrompue et le programme s'arrête.

— Lorsque le joueur a deviné la combinaison ou qu'il a épuisé ses 10 essais, le programme annonce que la partie est gagnée en un certain nombre de coups (M4) ou perdue (auquel cas il affiche la bonne combinaison) (M5), et demande ensuite au joueur s'il veut rejouer (Q2).

Si le joueur se trompe (c'est-à-dire qu'il entre une combinaison de couleurs non valide), l'ordinateur affiche un message d'erreur (M2). Dans ce cas, le compteur de coups n'est *pas* incrémenté (cf. l'exemple de partie en Section B).

Notez que le joueur doit pouvoir entrer une combinaison en lettres majuscules ou minuscules.

2 Consignes

Le fonctionnement de votre programme sera partiellement évalué par UpyLaB (Section 3). Ceci vous permettra de le valider de manière interactive, mais impose également de respecter une interface (input/output) très précise, décrite en Section A.

2.1 Remise du projet

Le projet est à réaliser **seul** et à remettre **au plus tard le vendredi 11 décembre 2015 à 12h30**. Nous vous demandons de vous limiter **uniquement** à la matière vue au cours des 8 premières séances de travaux pratiques.

Afin de vous aider à structurer votre approche, le projet fera l'objet de *remises partielles*. Suivant le calendrier ci-dessous, vous soumettrez plusieurs fonctions seules, qui seront validées par UpyLaB, avant de soumettre, toujours dans UpyLaB, votre programme complet. Les fonctions citées dans le calendrier sont décrites plus loin (Section 2.3).

Pratiquement, vous verrez dans UpyLaB des "exercices" correspondant aux différentes parties de code à soumettre. Après la date de remise, l'"*exercice*" disparaît d'UpyLaB, de sorte que vous ne puissiez plus modifier le code.

Calendrier de remise

vendredi 27/11/2015 la fonction `is_valid_pattern(pattern)`.

vendredi 4/12/2015 la fonction `try_pattern(pattern,code)`.

vendredi 11/12/2015 à 12h30 le programme complet.

Le code de votre projet sera évalué sur base de ce que vous aurez soumis dans UpyLaB pour le programme complet. Aucune autre forme de remise du projet n'est requise (ni permise) de votre part. Ceci implique que vous devez absolument respecter les délais impartis. Une fonction ou le code complet ne seront plus acceptés au-delà de l'échéance fixée.

Notez, toutefois, que vous pouvez "déposer" un code non complet dans UpyLaB au fur et à mesure de votre avancement. Ceci vous assure qu'en cas de problème technique, au moins une version (même s'il ne s'agit pas de la dernière) sera évaluée, plutôt que rien du tout.

2.2 Structures de données imposées

Etant donné la validation partielle par UpyLaB, nous devons également vous imposer le typage de certaines variables que vous utiliserez pour programmer le jeu. Le respect strict des structures de données suivantes est donc **impératif** :

`pattern` liste contenant une combinaison de 4 lettres, chaque élément de la liste étant un caractère unique. Exemple : `['j','r','b','c']` représente la combinaison de couleurs : *jaune, rouge, bleu, blanc*. Notez qu'un pattern *peut* contenir plusieurs fois le même caractère.

`code` pattern (cf. ci-dessus) contenant le code qui a été généré aléatoirement par le moteur de jeu et qui doit être trouvé par le joueur. Contrairement à `pattern`, un code ne peut pas contenir de doublons.

2.3 Découpe du code

On vous demande de découper votre programme en fonctions dont voici quelques prototypes à implémenter **obligatoirement** :

`play()` exécute le jeu. Cette fonction est celle qui est appelée par le programme pour le lancer. Elle ne reçoit aucun paramètre et ne renvoie rien.

`random_code()` génère de manière (pseudo-)aléatoire et renvoie un pattern, c'est à dire une liste de 4 lettres différentes prises parmi les 8 lettres possibles.

`is_valid_pattern(pattern)` vérifie le pattern donné en paramètre et renvoie `True` si la combinaison est valide, c'est à dire, qu'elle fait référence aux couleurs possibles et qu'elle a la bonne taille ; `False` est renvoyé si la combinaison n'est pas valide. [3]

`try_pattern(pattern,code)` évalue un pattern (une combinaison tentée par le joueur) par rapport au code à découvrir. Cette fonction renvoie un tuple, dont le premier élément représente le nombre de pions qui se trouvent à leur bonne place. Le deuxième élément représente le nombre de pions qui ont une "bonne" couleur par rapport au code, mais qui ne sont pas bien placés.

Notez qu'à l'exception de `play()`, aucune des fonctions présentées ci-dessus n'interagit avec l'utilisateur (ni `print`, ni `input`).

Ce découpage n'est pas complet et vous êtes invités à définir d'autres fonctions afin de rendre votre programme et vos fonctions le plus modulaire possible. Conformément aux règles de bonne pratique, chaque fonction doit avoir une tâche **unique** et bien définie et ne peut pas dépasser 15 lignes.

Vous devez gérer les saisies de l'utilisateur et les tester afin d'assurer la cohérence entre la donnée récupérée et le type de données requis. Par contre, les préconditions des fonctions, qui seront spécifiées dans les *docstrings*, ne doivent pas être explicitement vérifiées au début de vos fonctions. Ces conditions sont supposées remplies lors de l'utilisation de ces fonctions.

3. Nous vous conseillons de décomposer cette fonction en appelant au moins une autre fonction `is_valid_colors(pattern)` qui vérifie la validité des codes couleurs dans `pattern`.

3 Evaluation

L'évaluation du projet se fera en quatre étapes :

1. La première fonction intermédiaire est évaluée par UpyLaB.
2. La deuxième fonction intermédiaire est évaluée par UpyLaB.
3. Le programme complet est évalué par UpyLaB. Dès que vous soumettez un code (y compris durant le développement), UpyLaB vous donne le nombre de tests réussis. Vous pouvez en déduire la note partielle pour le fonctionnement du programme.
   Attention ! L'évaluation de votre programme par UpyLaB vous permet de le valider de manière interactive, mais impose également de respecter une interface (input/output) très précise, décrite en Section A.
4. Nous relisons le code du programme complet[4] pour en vérifier et évaluer la qualité, notamment en termes de structure (découpe, algorithmes, etc.) et de respect des règles de bonne pratique. Les critères généraux d'évaluation sont les suivants :
   — Le code **résout exactement le problème demandé sans ajout ni apport personnel**, qualité de l'algorithmique, qualité et pertinence de la découpe en fonction, syntaxe correcte, indentation parfaite, variables bien nommées.
   — Le code **respecte les conventions et les règles de bonne pratique** (disponibles sur l'Université Virtuelle, dans la section Projet). **En particulier, vous devez être attentif à ce que chaque fonction soit correctement documentée par une *docstring***.
   — Le respect de l'ensemble des consignes.

Plagiat. Votre code sera traité par un logiciel de détection de plagiat. Nous nous réservons la possibilité de convoquer les étudiants pour lesquels ce logiciel aurait signalé une suspicion de plagiat et/ou d'annuler la note globale du projet de l'ensemble des étudiants concernés.

4 Quelques recommandations

Nous vous conseillons vivement d'utiliser un environnement de développement (par exemple Idle) pour développer et tester votre programme. Ceci simplifiera significativement le déboguage de votre code durant la phase de développement.

Toutefois, étant donné que vous serez amenés à remettre votre programme dans UpyLaB, nous vous suggérons avec insistance de vérifier suffisamment tôt que le code qui fonctionne dans votre environnement de développement s'exécute également correctement dans UpyLaB. Vous pourrez notamment y vérifier si votre programme passe les tests prévus et connaître d'avance votre cote pour cette partie de l'évaluation. [5]

Attention : Seul le code des fonctions demandées sont à mettre dans UpyLaB lors des deux premières remises.

4. Etant donné que ce code contient les fonctions intermédiaires, celles-ci seront évaluées avec le programme complet. Nous ne relisons donc pas systématiquement les fonctions intermédiaires que vous avez remises dans UpyLaB.
5. Cette note partielle est définitive, sous réserve des vérifications de plagiat.

A Interface imposée pour UpyLaB

Votre programme sera validé partiellement par UpyLaB. Pour cette raison, il faudra que l'interfaçage, c'est à dire les `print` et `input` du programme respectent strictement une forme imposée. Cette section vous précise l'interface à respecter et qui sera utilisée par UpyLaB pour valider votre code. En cas de message d'erreur dans UpyLaB, veillez donc à vérifier si votre programme respecte l'interface. Notez que, pour être plus clair, tous les espaces sont représentés par un point (·). Lorsqu'il y a une option, la lettre en majuscule et en parenthèse (exemple : `(Q)uitter`) indique la réponse à donner par l'utilisateur (ici, `Q`). Dans les tableaux suivants, les paramètres N, X et Y sont des entiers (`int`) à remplacer par des valeurs adéquates.

Interactions

Ref	Contexte	Message
Q1	Introduction d'une nouvelle combinaison.	N`/10·>>>·`
Q2	Demande à l'utilisateur s'il veut jouer une nouvelle partie.	*(une ligne vide)* `Voulez-vous·[C]ontinuer·à·jouer·ou·[Q]uitter?`

Messages

Ref	Contexte	Message
M1	Message de démarrage d'une nouvelle partie.	*(une ligne vide)* `----< Nouvelle partie >----` *(une ligne vide)*
M2	Indique que la combinaison entrée n'est pas valide.	`Cette·combinaison·n'est·pas·valide.`
M3	Evalue une combinaison par rapport au code.	X`·pion(s)·en·place·et·`Y`·bonne(s)·couleur(s).`
M4	Annonce que la partie est gagnée.	`Bravo!·Vous·avez·remporté·la·partie·en·`N`·coups.`
M5	Annonce que la partie est perdue.	`Vous·avez·perdu·la·partie.·La·` `combinaison·à·trouver·était:·`$XXXX$`.`

Notez que les messages M2 à M5 sont précédés d'exactement 20 blancs (caractères d'espace), comme illustré en Section B.

Pour vous éviter de perdre inutilement de temps en mise en forme de l'interface du programme (messages affichés), nous vous proposons d'intégrer (copier/coller) les fonctions suivantes dans votre code et d'utiliser ces fonctions pour l'affichage des différents messages ainsi que pour les `input`. Le fichier Python `interface.py` correspondant à ces fonctions est disponible sur le l'UV, sous la rubrique *Projet*.

```
def print_msg_start():
    # Message M1
    """
    Affiche le message de début de partie.
    """
```

```
    print("\n----< Nouvelle partie >----\n")

def print_msg_wrong_entry():
    # Message M2
    """
    Affiche le message que la combinaison n'est pas valide.
    """
    print(20*" " + "Cette combinaison n'est pas valide.")

def print_msg_eval_entry(well_placed, good_colour):
    # Message M3
    """
    Affiche le nombre de pions biens placés [well_placed] et le
    nombre de pions de bonne couleur [good_colour].
    """
    print(20*" " + str(well_placed) + " pion(s) en place et " \
     + str(good_colour) + " bonne(s) couleur(s).")

def print_msg_game_won(count):
    # Message M4
    """
    Affiche que la partie a été gagnée en [count] coups.
    """
    print(20*" " + "Bravo! Vous avez remporté la partie en " + str(count) + " coups.");

def print_msg_game_lost(code):
    # Message M5
    """
    Affiche qua la partie a été perdue ainsi que le [code] qui était à trouver.
    """
    code_str = "".join(code)
    print(20*" " + "Vous avez perdu la partie. La combinaison à trouver était: " \
     + code_str + ".")



def input_prompt_entry(count):
    # Question Q1
    """
    Demande à l'utilisateur d'introduire sa combinaison pour le [count]-ième coup.
    """
    return input(str(count) + "/10 >>> ")

def input_continue_playing():
    # Question Q2
    """
    Demande à l'utilisateur s'il veut continuer à jouer ou quitter le programme.
    """
    return input("\nVoulez-vous [C]ontinuer à jouer ou [Q]uitter? ")
```

B Exemple de partie

Note Pour rendre cet exemple plus clair, nous avons souligné les *entrées* de l'utilisateur. Dans votre programme, il s'agira de texte "normal", sans mise en forme particulière.

```
----< Nouvelle partie >----

1/10 >>> RGBX
                    Cette combinaison n'est pas valide.
1/10 >>> RGBY
                    0 pion(s) en place et 3 bonne(s) couleur(s).
2/10 >>> GRBP
                    1 pion(s) en place et 2 bonne(s) couleur(s).
3/10 >>> GBRP
                    1 pion(s) en place et 2 bonne(s) couleur(s).
4/10 >>> GBPR
                    2 pion(s) en place et 1 bonne(s) couleur(s).
5/10 >>> GPRB
                    2 pion(s) en place et 1 bonne(s) couleur(s).
6/10 >>> GPBR
                    1 pion(s) en place et 2 bonne(s) couleur(s).
7/10 >>> GPCR
                    1 pion(s) en place et 1 bonne(s) couleur(s).
8/10 >>> GYPB
                    Bravo! Vous avez remporté la partie en 8 coups.

Voulez-vous [C]ontinuer à jouer ou [Q]uitter? c

----< Nouvelle partie >----

1/10 >>> RGBY
                    0 pion(s) en place et 3 bonne(s) couleur(s).
1/10 >>> …
```

Quelques exemples d'entrées erronées (les entrées de l'utilisateur sont soulignées)

```
---------------------------

1/10 >>> RGBX
                    Cette combinaison n'est pas valide.
1/10 >>> 12RG
                    Cette combinaison n'est pas valide.
1/10 >>> RGB
                    Cette combinaison n'est pas valide.
```