LMI 1 - Semestre 2 Mars-Avril 2015

Université d'Artois
Faculté des Sciences Jean Perrin

ALGO2 – Algorithmique et Programmation 2

Projet : Mastermind

Votre travail

Vous devez implémenter en `python` le jeu du Mastermind. Ce projet est à faire en binôme et doit être déposé sur Moodle rubrique `projet mastermind` pour le **15 avril 2015** dernier délai. Vous devez (chacun) donner le nom de votre binôme dans le questionnaire prévu à cet effet sur Moodle pour le **13 mars 2015 à 18h** dernier délai.

Vous serez évalués en fonction de la qualité de votre conception, votre implémentation, le bon fonctionnement de l'application et la documentation fournie. Vous veillerez en particulier à spécifier et à commenter vos classes, méthodes et fonctions.

Vous devrez fournir également un document descriptif des différentes fonctions utilisées. Ce document sera au format `pdf` et sera également déposé sur Moodle.

Attention : à la fin de chaque semaine, vous devrez déposer à l'emplacement réservé (”Dépôt Semaine2”, ”Dépôt Semaine3”, etc.) sur Moodle l'état d'avancement de votre application. Certains de ces fichiers rendus pourront être évalués. Vous déposerez également un document texte qui décrira votre application. Un contrôle individuel en rapport avec ce projet sera également effectué.

Le projet Mastermind

Le but de ce projet est de réaliser un jeu de mastermind, dont la version finale pourrait ressembler à la figure 1 :

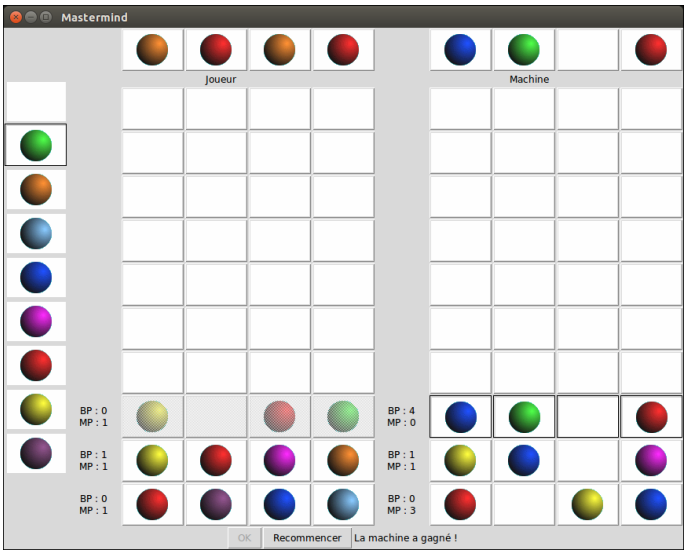

FIGURE 1 – Mastermind : interface graphique finale du projet

Le mastermind est un jeu de société qui se joue à deux. Ils disposent de pions colorés (nous considérerons les couleurs violet, jaune, rouge, orange, rose, bleu foncé, bleu clair, vert et le vide). Un joueur choisit un

code secret de quatre pions parmi ces couleurs. Le second joueur va essayer de trouver le code. Il a droit à dix essais. À chaque proposition qu'il fait, le premier joueur lui donne deux indications : le nombre de pions de la bonne couleur qui sont bien placés (BP), et le nombre de pions de la bonne couleur qui sont mal placés (MP).

Dans le version finale souhaitée, il se tiendra deux parties de mastermind en parallèle : celle du joueur, où l'ordinateur a choisi un code que le joueur humain essaie de trouver, et celle de l'ordinateur, où la machine essaie de trouver le code qu'elle-même a choisi (mais sans tricher !). Dans l'illustration précédente, l'ordinateur a trouvé le code secret en trois propositions seulement. La partie étant finie, les deux codes secrets à trouver ont été affichés tout en haut.

Ce projet a été découpé en six étapes, afin de vous accompagner dans sa réalisation. Chaque semaine, vous rendrez la nouvelle version de votre projet incluant les fonctionnalités demandées à chaque étape. Vous devez donc respecter l'ordre de réalisation pour chaque semaine. Une grande part de votre travail sera de respecter la bonne décomposition du projet : la séparation du moteur du jeu (le modèle, qui fait les calculs) de la partie qui gère l'interface graphique, . . .Vous serez guidé dans cette décomposition, qui vous permettra d'avancer facilement d'étape en étape.

Planning du projet

Semaine 1 - le modèle (1)

Pour la semaine 1 et la semaine 2, vous devrez écrire des tests unitaires pour vos méthodes.

La première semaine sera consacrée à réaliser une première version en mode texte du jeu.
Exemple d'interaction (ici, on joue avec 6 couleurs de 0 à 5) :

```
projet-mastermind$ python3 main_mastermind.py
Tour 1
Votre proposition ? [1,2,3,4]
La réponse est  0 bien placés et  1 mal placés.
Tour 2
Votre proposition ? [0,1,5,0]
La réponse est  1 bien placés et  2 mal placés.
Tour 3
Votre proposition ? [0,5,0,2]
La réponse est  1 bien placés et  1 mal placés.
Tour 4
Votre proposition ? [0,0,5,1]
La réponse est  0 bien placés et  3 mal placés.
Tour 5
Votre proposition ? [5,1,0,5]
La réponse est  4 bien placés et  0 mal placés.
Bravo, la partie est finie, vous avez gagné.
```

Vous devez programmer la classe `Proposition`. Elle contient un code (une liste de quatre valeurs), et propose un certain nombre de méthodes pour le manipuler :

- la méthode constructeur prend en paramètre une liste d'entiers qui représentera le code de la proposition.
- `calcule_bp_mp(self,code)` : cette méthode prend en paramètre un `code` numérique sous la forme d'une liste d'entiers et retourne le nombre de bien placés et le nombre de mal placés dans `code` par rapport à `self`. La spécification de cette méthode est `Proposition, list(int)`

`-> int, int`. C'est la première fonction très importante pour cette première semaine (et même pour tout le projet). Nous reviendrons ensuite sur cette méthode.

— `__str__(self)` : une méthode pour afficher une `Proposition`.

Cette classe doit être compatible avec le code suivant :

```
>>> from mastermind import Proposition
>>> p = Proposition([1,2,3,4])
>>> print(p)
[1, 2, 3, 4]
>>> p.calcule_bp_mp([3,2,3,1])
(2, 1)
>>> print(p)
[1, 2, 3, 4]
>>> p = Proposition([1,1,3,4])
>>> p.calcule_bp_mp([3,2,3,1])
(1, 1)
```

Vous aurez besoin de calculer dans un premier temps les bien placés, puis dans un deuxième temps les mal placés. Attention : une valeur ne doit pas être utilisée deux fois. Dans le premier exemple ci-dessus, le `3` qui est dans `p` est utilisée pour compter un bien placé, mais ne doit pas être utilisé ensuite lorsqu'on va compter les mal placés. Dans le deuxième exemple ci-dessus, le dernier `1` du code passé en paramètre de la méthode `calcule_bp_mp` ne compte qu'une seule fois comme un mal placé. Il faut donc retirer au fur et à mesure les valeurs qui sont comptabilisées (en bien placé comme en mal placé). Pour cela, vous prendrez soin au début de votre méthode de faire en sorte de travailler sur des listes d'entier qui soient des copies du contenu de la proposition comme du code passé en paramètre.

Vous devrez également programmer la classe `Mastermind` qui contient la modélisation du jeu général. Elle permettra ultérieurement de gérer une partie où c'est l'ordinateur qui 'joue'. Pour le moment, la classe `Mastermind` contient :

— le constructeur qui prend en paramètre le nombre de couleurs (i.e. si le nombre de couleurs est `6`, alors les valeurs d'un code seront comprises entre `0` et `5`) et la taille d'un code (nombre de valeurs dans un code). Une valeur par défaut sera donnée pour ces deux paramètres. Le constructeur initialise également un attribut `__mystere` à `None`

— une méthode `dim(self)` qui retourne la taille du code

— une méthode `lancer(self)` qui génére aléatoirement un code secret. Ce code secret sera une `Proposition` qui sera mémorisé par l'attribut `__mystere`.

— une méthode `bp_mp(self,code)` dont la spécification est `Mastermind, list(int) -> int,int` et qui retourne le nombre de bien placés et de mal placés dans le code passé en paramètre par rapport au mystère du `Mastermind`.

Dans cette première étape, vous devrez implémenter les deux classes `Proposition` et `Mastermind`. Elle devront être compatible avec le code suivant pour le jeu en mode texte :

```
import mastermind

mm = mastermind.Mastermind()
## pour générer un code secret :
mm.lancer()
cpt = 1
bp, mp = 0, 0
while (cpt < 10) and (bp != mm.dim()) :
    print("Tour "+str(cpt))
```

```
    rep = eval(input("Votre proposition ? "))
    bp,mp = mm.bp_mp(rep)
    print("La réponse est ",bp,"bien placés et ",mp,"mal placés.")
    cpt += 1
## end of while
if (bp,mp) == (mm.dim(),0) :
    print("Bravo, la partie est finie, vous avez gagné.")
```

Ces contraintes sont impératives, et vous garantiront une modélisation correcte et facilement maintenable de votre application lorsque vous aborderez la partie graphique.

Semaine 2 - le modèle (2)

Pour la semaine 1 et la semaine 2, vous devrez écrire des tests unitaires pour vos méthodes.

Dans cette deuxième semaine, nous restons sur le mastermind en mode texte mais nous allons réaliser la partie où c'est l'ordinateur qui cherche à deviner le code secret. Pour cela, vous allez enrichir les deux classes `Proposition` et `Mastermind`.

L'algorithme pour jouer est simple. Vous remarquerez d'abord qu'on peut énumérer toutes les combinaisons. Pour une taille de `4` et un nombre de couleurs de `6`, les combinaisons vont de `[0,0,0,0]` à `[5,5,5,5]`. On passe d'une proposition à la suivante en ajoutant `1` au chiffre le plus à gauche (ou à droite, peu importe, choisissez votre convention). Si le chiffre obtenu atteint le nombre de couleurs du mastermind, alors on le passe à zéro et on reporte la retenue sur le chiffre immédiatement à droite. La méthode `proposition_suivante(self)` de la classe `Proposition` retourne une nouvelle proposition qui est la combinaison suivante de `self`.

```
>>> p = Proposition([0,0,0,0])
>>> p2 = p.proposition_suivante()
>>> print(p2)
[1, 0, 0, 0]
>>> p = Proposition([1,2,3,4])
>>> p2 = p.proposition_suivante()
>>> print(p2)
[2, 2, 3, 4]
>>> p = Proposition([5,2,3,4])
>>> p2 = p.proposition_suivante()
>>> print(p2)
[0, 3, 3, 4]
>>> p = Proposition([5,5,5,5])
>>> p2 = p.proposition_suivante()
>>> print(p2)
[0, 0, 0, 0]
```

Pour jouer, une idée simple (qui permet de trouver assez rapidement la solution !) est donc d'énumérer toutes les combinaisons, et de s'arrêter lorsqu'on trouve une combinaison qui pourrait être le code secret : si c'était elle le code secret, elle aurait retourné le même nombre de pions bien placés et mal placés pour chaque proposition déjà faite que ce qui a été retourné. C'est cette proposition qui est jouée par l'ordinateur.

La classe `Proposition` doit maintenant posséder deux attributs `__bp` et `__mp` qui sont initialisés à zéro. La classe `Proposition` possède les nouvelles méthodes suivantes :

— `set_bp(self,bp)` qui permet d'affecter une valeur à l'attribut `__bp`

— `set_mp(self,mp)` qui permet d'affecter une valeur à l'attribut `__mp`
— `bp(self)`, `mp(self)` et `code(self)` qui retournent respectivement la valeur de `__bp`, `__mp` et une copie du code contenu dans la `Proposition`
— `proposition_suivante(self)` qui a déjà été discutée
— `valide(self,code)` dont la spécification est `Proposition,list(int) -> boolean` qui vérifie que les nombres de bien placés et de mal placés de `code` par rapport à `self` sont bien les mêmes que ceux mémorisés dans `__bp` et `__mp`.
— et toujours la méthode `__str__(self)` que vous n'oublierez pas de mettre à jour (on veut maintenant afficher le nombre de bien et de mal placés.

La classe `Mastermind` doit maintenant proposer un nouvel attribut qui est l'historique des propositions déjà faites. Elle possède également :

— une méthode `est_valide(self,code)` qui vérifie si `code` est valide pour chaque proposition de l'historique.
— une méthode `proposition(self)` qui retourne la prochaine combinaison valide en fonction de l'historique des propositions déjà faites. Si aucune proposition n'a déjà été faite, alors la première proposition est générée aléatoirement. Cette proposition est ajoutée à l'historique.
— une méthode `prend_score(self,bp,mp)` qui affecte le score de bien placés (`bp`) et de mal placés (`mp`) à la dernière proposition de l'historique.
— une méthode `partie_finie(self)` qui teste si la partie est finie (soit parce que l'ordinateur a gagné, soit parce qu'il a épuisé tous les essais permis).
— une méthode `__str__(self)` qui présente tout l'historique des propositions.

Enfin, vous adapterez le script principal proposé pour la semaine 1 afin d'être compatible avec la trace suivante. Vous ferez bien attention qu'il y a maintenant deux parties de mastermind en cours en même temps :

```
projet-mastermind$ python3 main_mastermind.py
Tour 1
Votre proposition ? [0,1,2,3]
La réponse est  1 bien placés et  2 mal placés.
Ma proposition est : [0, 4, 2, 0]
Mon score est : 0  bien placés,  2  mal placés.
Tour 2
Votre proposition ? [0,2,1,4]
La réponse est  1 bien placés et  1 mal placés.
Ma proposition est : [1, 0, 0, 1]
Mon score est : 1  bien placés,  0  mal placés.
Tour 3
Votre proposition ? [0,5,5,1]
La réponse est  0 bien placés et  1 mal placés.
Ma proposition est : [4, 2, 3, 1]
Mon score est : 2  bien placés,  2  mal placés.
Tour 4
Votre proposition ? [3,2,0,3]
La réponse est  1 bien placés et  1 mal placés.
Ma proposition est : [3, 2, 4, 1]
Mon score est : 1  bien placés,  3  mal placés.
Tour 5
Votre proposition ? [3,1,1,2]
La réponse est  4 bien placés et  0 mal placés.
Bravo, la partie est finie, vous avez gagné.
```

Semaine 3 - la vue

Vous allez commencer la réalisation de l'interface graphique. Dans un premier temps, votre mastermind pourrait ressembler à la figure 2. Toutes les images sont disponibles sur Moodle.

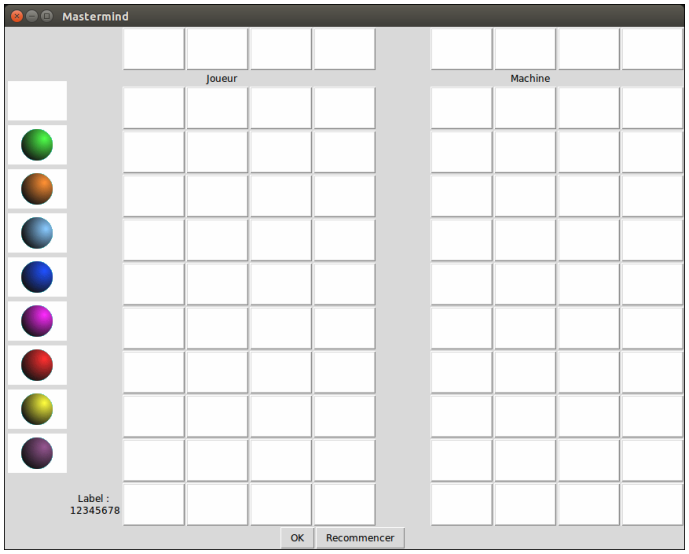

FIGURE 2 – Mastermind : une première interface graphique

— Créez la vue dans la classe `VueMastermind` et tous les composants nécessaires pour ressembler à la 2. Les emplacements blancs sont des boutons qui affichent l'image `sphere0.gif` (les autres images sont également posées sur des boutons). À gauche des emplacements, on trouve des étiquettes de largeur 8 qui serviront à afficher le score de chaque proposition.
— Faites en sorte que l'appui sur une des couleurs sur le coté gauche met le bouton en évidence (vous pouvez jouer sur les propriétés de `relief` des boutons). Lorsqu'on change de couleur, la première redevient normale, et la nouvelle est mise en évidence.
— Faites en sorte que le choix d'une couleur mette à jour un indice qui corresponde à la couleur choisie. Réfléchissez à ce qu'il faut ajouter à `VueMastermind`.
— Faites en sorte que seuls les boutons de la dernière ligne du joueur soient actifs. Un clic sur un bouton de cette ligne lui fait prendre la couleur sélectionnée.
— Faites en sorte qu'un clic sur le bouton `Ok` : ne permette plus de modifier la ligne de pions qui vient d'être fixée ; rende les boutons de la ligne immédiatement au-dessus actifs (ce sont alors eux qui peuvent recevoir les couleurs).
— Créez un script principal qui lance la vue.

Semaine 4 - les contrôleurs (1)

L'objectif cette semaine est de relier le modèle et la vue pour la partie du joueur (partie gauche du plateau).

— Dans le schéma MVC, la vue doit toujours connaître le modèle. Modifiez en conséquence le constructeur de `VueMastermind` et le script principal. Vous n'oublierez pas que (pour le moment au moins), le nombre de couleurs est fixé à 9, et la dimension du code à 4.
— Modifiez la classe `Mastermind` pour permettre de recommencer une partie.
— Faites en sorte qu'un clic sur le bouton `Recommencer` appelle une fonction contrôleur qui permette de réinitialiser une partie : cela concerne aussi bien le modèle que la vue.
— Enfin, réfléchissez à la dernière question pour cette semaine : lorsque l'utilisateur clique sur le bouton `Ok`, il faut dialoguer avec le modèle pour obtenir le nombre de bien placés et le nombre

de mal placés dans la proposition du joueur. Quelle est la modification que vous devez apporter à `VueMastermind` pour être en capacité de transmettre une information au modèle ? Faites maintenant en sorte que le bouton `Ok` appelle un contrôleur qui asure ce dialogue et les mises à jour de part et d'autre.

Semaine 5 - les contrôleurs (2)

L'objectif de cette semaine est de faire jouer également la machine, et de prendre en compte la fin de partie. Vous allez enfin faire le lien avec le `mastermind` tel que vous l'avez développé en semaine 2.

— Lorsque le joueur clique sur `Ok`, il soumet sa proposition et obtient l'information sur le nombre de bien placés et de mal placés. À ce moment-là (et si le joueur ne vient pas de gagner la partie), c'est à l'ordinateur de faire une proposition, et d'obtenir une réponse sur son nombre de bien et de mal placés. Implémentez cette fonctionnalité. Vous devrez modifier le script principal, la classe `VueMastermind`, et les contrôleurs associés aux boutons `Ok` et `Recommencer`.

— Dès que la partie se termine, soit parce qu'un des deux joueurs a gagné, soit parce qu'ils ont épuisé le nombre d'essais autorisés, les deux codes secrets doivent apparaître en haut de la fenêtre et il ne doit plus être possible de cliquer sur un bouton autre que le bouton `Recommencer`. Implémentez cette fonctionnalité. Vous devrez modifier la classe `Mastermind` pour qu'elle vous retourne son code secret.

Semaine 6 - derniers ajustements - améliorations

Cett dernière semaine vous permettra de terminer les fonctionnalités qui vous manquent, de peaufiner votre code (améliorer la conception, les algorithmes), de soigner votre documentation...

Vous pourrez aussi apporter des améliorations : permettre par exemple de s'adapter à un nombre de couleurs ou à une taille de code choisi dans le script principal, même si on limite le nombre de couleurs à 9.