Travail pratique # 3
PyMineur

(en équipes de 2)

Simon Hardy

Date de remise : le jeudi 24 mars 2016
Pondération de la note finale : 12%

1 Objectifs

Ce travail a comme principal objectif de vous familiariser davantage avec la programmation orientée objet, via une modélisation déjà faite pour vous, que vous devez d'abord comprendre puis compléter. Ce sera également l'occasion d'utiliser la structure de données « dictionnaire ». Ce travail vous permettra de valider votre compréhension de la matière des modules 1 à 6, inclusivement. La modélisation que nous vous fournissons prend pour acquis que vous comprenez très bien le principe de la décomposition fonctionnelle et la réutilisation de fonctions. Souvent, une méthode pourra être programmée en trois ou quatre lignes de code, lorsqu'on réutilise les méthodes programmées préalablement.

2 Organisation du travail en équipe

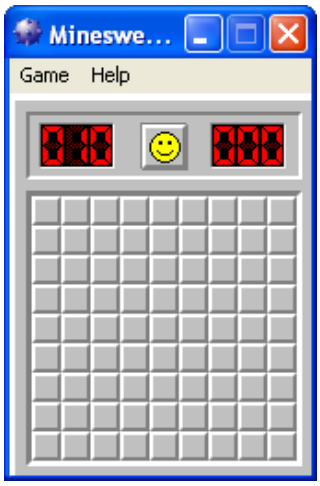
Nous vous demandons de travailler en équipe car c'est un objectif de votre formation académique, et vous pourrez ainsi vous partager la charge de travail. Pour ceux qui n'ont pas encore trouvé de coéquipier, nous vous invitons à utiliser le forum du cours dans la section prévue à cet effet. Chaque coéquipier doit contribuer à part égale au développement de ce travail. Laisser son coéquipier faire tout le travail (peu importe les raisons) est inacceptable : vous passerez à côté des objectifs de ce cours. De la même manière, il ne faut pas non plus trop en faire : il faut apprendre à travailler en équipe ! Chaque équipe doit être inscrite sur le portail des cours, sur la page du TP3.

Nous vous fournissons sur le site Web du cours un tutoriel pour utiliser des technologies permettant de partager votre code avec votre coéquipier. Chaque membre d'une équipe possède sur son ordinateur une copie locale du code, mais le code est synchronisé par un service dans le « cloud » et nous pouvons également voir l'historique (qui a fait quoi et

quand). De plus, PyCharm intègre ces outils de manière très conviviale. Des démonstrations ont été faites lors des travaux dirigés.

3 Le problème à résoudre

Nous vous proposons de programmer le jeu Démineur en mode console, un jeu populaire fourni avec le système d'exploitation Windows.



Pour bien comprendre le jeu et avant toute programmation, vous devez y jouer au moins cinq fois. Vous pouvez jouer en ligne sur ce site : http://minesweeperonline.com/#beginner

La modélisation orientée objet pour ce TP est fournie, dans un package nommé `pymineur`. Un package est un dossier tout à fait standard, contenant des modules Python, et un module spécial nommé `__init__.py`. Importer des modules qui sont situés dans un package se fait de la manière suivante :

```
from nom_package.nom_module import nom_fonction
from nom_package.nom_module import NomClasse
```

Nous vous rappelons finalement que pour qu'un package soit correctement reconnu par PyCharm, vous devez créer un projet (ou ouvrir le dossier) contenant ce package. Ouvrir les fichiers `.py` individuellement ne fonctionnera pas.

3.1 Les règles du jeu

Le jeu se joue avec un tableau de cases dont le contenu est caché. Ces cases contiennent soit :

- Une mine
- Une case vide

- Un nombre qui indique combien de mines la case a dans son voisinage (les huit cases sur son pourtour). Notez qu'une case vide peut être représentée par le chiffre '0'.

Les mines sont placées aléatoirement dans le tableau.

Les règles du jeu sont les suivantes :

1. Si le joueur choisit une case où une mine est cachée, la mine explose ! La partie est terminée.
2. Si le joueur choisit une case avec un nombre caché, la case est dévoilée et le nombre devient visible.
3. Si le joueur choisit une case vide (donc qui n'a ni mine ni nombre caché), il y a un effet en cascade (voir section plus bas) qui fait le dévoilement de toutes les cases vides dans le voisinage jusqu'à ce que la limite du tableau soit atteinte ou qu'une case avec un numéro caché soit atteinte.

L'objectif du jeu est d'identifier, par la logique, toutes les cases contenant des mines, sans en déclencher aucune.

4 Spécifications du programme

1. Pour réduire la complexité du programme, la taille du tableau doit initialement être fixe avec 5 rangées et 5 colonnes. Le nombre de mines cachées doit être de 5.
2. Dans le tableau, les cases vides peuvent être représentées pas le nombre '0' ; ce qui indique que la case n'a aucune mine dans son voisinage. Le voisinage d'une case correspond à toutes les cases immédiatement adjacentes à la case horizontalement, verticalement et dans les diagonales.
3. Le programme devrait vérifier toutes les erreurs d'input de l'utilisateur. Deux erreurs de base : l'utilisateur entre des coordonnées qui ne sont pas des entiers pour les numéros de rangée et de colonne ; l'usager choisit des coordonnées qui ne sont pas dans le tableau. En cas d'input erroné, le programme demande à utilisateur d'entrer de nouvelles coordonnées. Vous penserez peut-être à d'autres types d'erreur.
4. Si le joueur entre des coordonnées pour une case déjà dévoilée, informez-le et demandez-lui de nouvelles coordonnées.
5. À la fin de la partie, on informe le joueur s'il a gagné ou perdu et on affiche un tableau solution où toutes les cases sont dévoilées.

4.1 La classe `Partie`

Le module `partie.py` contient une classe nommée `Partie`, modélisant une partie de démineur avec ses données et ses traitements. Cette classe manipule un objet de la classe `Tableau` qui est un attribut. Les méthodes de cette classe permettent de demander à l'utilisateur les coordonnées de la case à dévoiler, puis appelle les méthodes de la classe `Tableau`

pour valider les coordonnées, dévoiler la case et afficher le tableau. À chaque dévoilement, cette classe vérifie si le jeu est terminé ou s'il peut se poursuivre. Consultez le contenu du module `partie.py` pour y retrouver toutes les informations pertinentes.

4.2 La classe `Tableau`

La classe `Tableau` modélise les données et les traitements d'un tableau du jeu démineur, ayant comme principal attribut un dictionnaire de cases.

Les clés de ce dictionnaire sont des tuples (x, y) représentant les coordonnées d'une case. Les coordonnées d'une case dans le tableau sont composées de deux entiers : le premier nombre correspondant à la rangée, puis le deuxième nombre correspondant à la colonne. Les éléments de ce dictionnaire sont des objets de la classe `Case`.

Les méthodes de cette classe permettent entre autres d'initialiser le tableau en y plaçant les mines, de procéder à diverses validations de coordonnées, d'afficher le tableau et la solution, puis de dévoiler des cases en exécutant l'effet en cascade de dévoilement si nécessaire. Nous vous invitons à consulter le module `tableau.py` pour y retrouver toutes les informations pertinentes.

4.3 La classe `Case`

La classe `Case` modélise les données et les traitements d'une case d'un tableau du jeu démineur. Cette classe est relativement simple et les méthodes à programmer tiennent toutes en une ligne chacune. La classe a trois attributs qui spécifient si la case est minée, si elle a été dévoilée et quel est son nombre caché. Consultez le contenu du module `case.py` pour y retrouver toutes les informations pertinentes.

4.4 Le module `__main__.py`

Lorsque nous travaillons avec un package (composé de modules), il est possible de créer un module nommé `__main__.py`, qui sera le point d'entrée du package, c'est à dire qui sera exécuté si un utilisateur exécute directement votre package. C'est ce module qui contient le code pour jouer une partie de démineur. On instancie simplement un objet de type Partie, puis on appelle la méthode `jouer()`.

4.5 Algorithme de haut-niveau

1. Dans le module `__main__.py` du package, vous créez une objet de la classe Partie. Dans le constructeur de Partie, on crée un objet tableau_mines de la classe Tableau, qui est un attribut. Lors de l'instanciation de tableau_mine, celui-ci est initialisé. Il s'agit d'un dictionnaire qui contient les cases du tableau et dont les clés sont les coordonnées des cases.

2. Lors de l'initialisation du tableau, placez-y les mines aléatoirement. Lorsque vous placez une mine dans une case, incrémentez le nombre caché des cases voisines. Vous devriez obtenir quelque chose qui ressemble à ça si vous l'affichez avec une méthode comme afficher_solution() (si celle-ci n'affiche pas les numéros de rangées et de colonnes par exemple) :

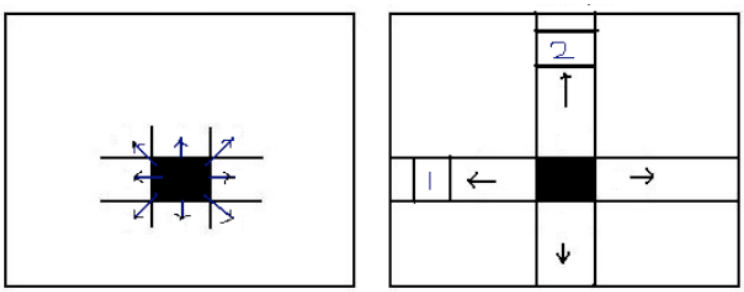
0	1	1	1	1
1	M	1	2	M
3	3	2	2	M
M	M	1	1	1
2	2	1	0	0

3. Affichez le tableau pour le joueur où les cases qui n'ont pas été dévoilées apparaissent comme étant cachées. Au début du jeu, toutes les cases devraient être cachées. Au fur et à mesure que le joueur choisit de dévoiler des cases, la vue du joueur doit être mise à jour.
4. Demandez au joueur d'entrer les coordonnées d'une case (numéros de rangée et de colonne) du tableau qu'il ou elle veut dévoiler. Selon les dimensions initiales, les rangées et les colonnes sont numérotées de 1 à 5.
5. Si le joueur dévoile une mine, informez-le que c'est une mine, montrez la solution et la partie se termine.
6. Si le joueur dévoile un numéro, montrez le tableau mis à jour.
7. Si le joueur dévoile une case vide, montrez le tableau après l'effet en cascade.
8. Répétez les étapes 3 à 7 tant que la partie n'est pas finie. La partie se termine si toutes les cases ne contenant pas de mines ont été dévoilées ou si une mine est déclenchée.

4.6 Effet de dévoilement en cascade

Cas 1 - Exigence minimale pour votre tp Si le joueur choisit une case vide dans le tableau, considérez uniquement la rangée et la colonne de la case. En partant de la case, dévoilez les cases à l'horizontale et à la verticale jusqu'à ce que vous rencontriez une case avec un nombre caché, ou la limite du tableau le cas échéant. Rencontrer une case avec un nombre caché provoque l'arrêt de l'effet en cascade dans cette direction. C'est le cas de droite dans l'image ci-dessous où la case en noir est celle choisie par le joueur.

Cas 2 - Exigence facultative pour votre tp Dans le cas idéal, toutes les 8 cases voisines devraient être considérées. Si une de ces cases voisines est aussi vide, alors ses 8 voisins à elle devraient alors être considérés pour un dévoilement, et ainsi de suite. Bien sûr, les limites du tableau sont aussi à considérer. L'implémentation de cet effet en cascade demande l'utilisation de récursivité ou d'une file. Comme nous n'avons pas vu ça, ce n'est pas obligatoire. Mais ceux qui veulent un défi peuvent essayer. C'est le cas de gauche dans l'image ci-dessous.



5 Comment attaquer le problème

Le problème à résoudre étant plutôt grand, commencez par bien comprendre la modélisation fournie. Lisez toute la documentation des diverses classes et méthodes, et réfléchissez à la manière dont vous pourrez résoudre les problèmes. Déjà avec le nom et la documentation des méthodes, vous devriez être en mesure de vous dire « pour programmer cette méthode, je ferai probablement appel à telle et telle méthode ».

Lorsque vous programmez une méthode, assurez-vous de bien regarder quels outils sont à votre disposition. Qu'avez-vous programmé précédemment ? Comment pouvez-vous réutiliser ces méthodes ? Il est très important de programmer et tester au fur et à mesure. N'essayez surtout pas de tout programmer sans tester, vous n'y arriverez pas.

6 Les méthodes à programmer

Vous devez programmer toutes les méthodes qui ne sont pas déjà programmées. Chacune de ces méthodes possède un commentaire `# TODO`, qui vous indique que vous devez programmer la méthode. Attention de ne pas en oublier !

Si vous comprenez bien l'interaction entre les diverses méthodes, chacune d'entre elles devrait être utilisée. Nous vous suggérons de tester vos méthodes à l'aide de tests unitaires. Nous vous fournissons un répertoire avec des exemples que vous pouvez compléter. Ceci dit, les tests unitaires ne seront pas corrigés et ils ne doivent pas être remis.

6.1 Fonctionnalités facultatives

Pour les équipes qui veulent en faire plus, vous pouvez bien sûr programmer l'effet en cascade qui considère tous les voisins, mais vous pouvez aussi permettre à l'utilisateur de spécifier un tableau de la taille de son choix avec le nombre de mines désiré. Le vrai jeu de démineur permet aussi au joueur de placer des drapeaux sur une case non-dévoilée lorsqu'il détermine qu'une mine s'y cache. Vous pouvez ajouter cette fonctionnalité, et d'autres à votre guise (voir prochaine section sur le prix Pierre Ardouin).

7 Le prix Pierre Ardouin

Depuis l'automne 2013, le département d'informatique et de génie logiciel a mis en place un concours récompensant l'équipe qui aura produit le meilleur TP/projet dans le cadre d'un cours. Ces travaux de session ont l'envergure d'un mini-projet qui est admissible par rapport aux normes fixées par le département. À la suite des évaluations des travaux, l'enseignant du cours détermine l'équipe gagnante ; chaque membre de l'équipe gagnante reçoit alors une bourse de 50$ ainsi qu'une attestation remise par le département.

De plus, le département d'informatique et de génie logiciel a mis en place une bourse Élite, appelée bourse « Pierre Ardouin », qui vise à récompenser le meilleur projet de session, tous cours confondus. Deux principaux critères guident le choix des évaluateurs dans l'identification du lauréat : l'excellence du travail (par rapport à ce qui est demandé dans l'énoncé) et l'aspect créativité/innovation. Il est actuellement prévu une bourse de 200$ pour récompenser chaque membre de l'équipe « élite » gagnante (pour un maximum de 1000$ pour toute l'équipe). Aussi, le département veille à publier l'information sur un site Web dédié : http ://www.ift.ulaval.ca/vie-etudiante/prix-pierre-ardouin.

À la deuxième moitié du mois de mai de chaque année universitaire, le département organise une cérémonie pour honorer les finalistes et le lauréat du prix « Pierre Ardouin » des sessions d'automne et d'hiver, et leur remettre une attestation.

Le travail pratique 4 sera une continuité du travail pratique 3, et l'ensemble de ces deux travaux sera évalué pour le prix Pierre Ardouin. Notez que l'aspect créativité/innovation sera évaluée sur les éléments du TP4, qui contiendra une interface graphique.

8 Modalités d'évaluation

Ce travail sera évalué sur 12 points. Voici la grille de correction :

Élément	Pondération
Le programme initialise correctement le tableau de mines	2 points
Le programme affiche le tableau selon l'état de dévoilement des cases	1 point
Le programme valide les coordonnées	2 points
Le programme implémente l'effet de dévoilement en cascade (de base)	1 point
Le programme permet un bon déroulement de la partie	1 point
La partie se termine si le joueur déclenche une mine	1 point
La partie se termine si le joueur a dévoilé toutes les cases sans mine	1 point
La solution s'affiche à la fin de la partie	1 point
Qualité du code (noms de variables, style, commentaires, documentation)	2 points

Notez qu'un programme qui n'est pas fonctionnel (qui ne s'exécute pas ou qui plante à l'exécution) pourrait recevoir une note de 0.

9 Remarques

Plagiat : Tel que décrit dans le plan de cours, le plagiat est strictement interdit. Ne partagez pas votre code source à quiconque. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toute circonstance. Tous les cas détectés seront référés à la direction de la faculté. Des logiciels comparant chaque paire de TPs pourraient être utilisés pour détecter les cas de plagiat.

Retards : Tout travail remis en retard peut être envoyé par courriel à l'enseignant si le portail des cours n'accepte pas la remise. Une pénalité de 25% sera appliquée pour un travail remis le lendemain de la remise. Une note de 0 sera attribuée pour un plus long retard.

Remises multiples : Il vous est possible de remettre votre TP plusieurs fois sur le portail des cours, en conservant le même nom de fichier. La dernière version sera considérée pour la correction. Ne laissez pas plusieurs fichiers avec des noms différents sur le portail.

Respect des normes de programmation : Nous vous demandons de prêter attention au respect des normes de programmation établies pour le langage Python, notamment de nommer vos variables et fonctions en utilisant la convention suivante : `ma_variable`, `fichier_entree`, etc. Utiliser PyCharm s'avère être une très bonne idée ici, car celui-ci nous donne des indications sur la qualité de notre code (en marge à droite, et souligné).

10 Ce que vous devez rendre

Votre programme doit être rédigé à même les fichiers Python fournis, que vous devez compresser dans une archive `.zip`. Ce TP étant développé dans un package, il vous faut zipper le dossier dans lequel se trouve votre package. Vous devez remettre donc une archive `.zip` d'un dossier, contenant :

- `correction.txt`
- dossier `pymineur`
  - `__init__.py`
  - `__main__.py`
  - `partie.py`
  - `tableau.py`
  - `case.py`

Vous n'avez pas à remettre le dossier de tests unitaires. Cette archive doit être remise via le site Web du cours.

Bon travail !