Travail pratique # 3
Jeu du démineur

Date de remise : au plus tard le dimanche 21 décembre 2025 à 23h59
Pondération de la note finale : 10 %

Un dernier TP pour la route ! C'est en pratiquant qu'on s'améliore. Chaque bogue élucidé est un pas vers le prochain bogue.

1 Objectifs

Ce travail vous permettra de vous familiariser davantage avec les structures de données principales de Python, comme les listes et les dictionnaires, ainsi que la modélisation de programmes informatiques via le paradigme orienté objet. Ce sera aussi l'occasion de vous familiariser avec la création d'interfaces graphiques. La majeure partie de la modélisation du problème est déjà faite pour vous. **Vous devez prendre le temps de bien lire la documentation fournie dans les modules afin de découvrir les liens entre les diverses classes que vous devez compléter.**

Finalement, ce travail vous permettra de valider votre compréhension de la matière des modules 1 à 8, inclusivement. La modélisation que nous vous fournissons tient pour acquis que vous comprenez bien le principe de la décomposition fonctionnelle et la réutilisation de fonctions. Souvent une méthode peut être programmée plus simplement lorsqu'on réutilise les méthodes déjà programmées préalablement.

2 Organisation du travail en équipe

Nous vous suggérons de travailler en **équipe de deux**, car c'est un objectif de votre formation universitaire, et vous pourrez ainsi vous partager la charge de travail. **Chaque équipe doit être formée sur le portail des cours, sur la page du TP3**. Pour ceux et celles 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 parts égales 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 !

Notez que l'historique des dépôts GitHub permet de voir quel membre de l'équipe est responsable de chacun des *commits*.

3 Le problème à résoudre

Vous devez créer une version console et une version interface graphique (*avec la librairie Tkinter*) du jeu démineur. C'est un jeu très populaire à l'époque puisqu'il était fourni avec le système d'exploitation Windows [1].

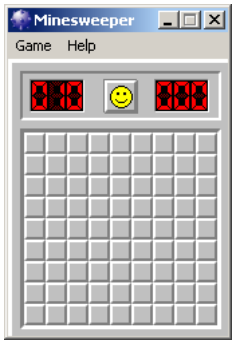

FIG. 1 : *
Une partie 8x8 pas encore commencée

FIG. 2 : *
Une partie 8x8 perdue

FIG. 3 : *
Une partie 8x8, avec 10 mines, réussie

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

La modélisation du jeu dans le paradigme orienté objet, ainsi qu'une grande partie de l'interface graphique, sont déjà réalisées pour vous. Votre tâche consiste à compléter certaines méthodes, qui sont déjà documentées, en suivant scrupuleusement cette modélisation. Il est impératif de respecter cette modélisation et de ne pas modifier le comportement des méthodes existantes. En revanche, vous êtes libre de créer de nouvelles méthodes si nécessaire, à condition de bien les documenter.

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
- Rien
- Un nombre qui indique combien de mines la case a dans son voisinage (les huit cases sur son pourtour). Notez qu'une case vide est simplement une case n'ayant ni mine, ni aucune mine dans son voisinage.

[1] Microsoft a cessé de fournir le jeu Démineur avec les installations par défaut de leur système d'exploitation à partir de Windows 8. Pour la petite histoire, voir https ://en.wikipedia.org/wiki/Microsoft_Minesweeper (en anglais)

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. En d'autres termes, toutes les cases *sauf* celles contenant des mines doivent être dévoilées.

4 Ordre de travail recommandé

Important : Nous vous recommandons fortement de procéder dans l'ordre suivant :

1. Commencez par le mode console : ce mode vous permettra de tester et valider toutes les méthodes fondamentales du jeu (classes `Tableau`, `Case`, `Partie`).
2. Utilisez les tests unitaires fournis pour valider vos méthodes au fur et à mesure. Les tests se trouvent dans les fichiers `test_case.py` et `test_tableau.py` et peuvent être exécutés directement.
3. Une fois le mode console fonctionnel, passez au mode graphique en exécutant le fichier `demineur_graphique.py`.

5 Spécifications du jeu en mode console

- **Pour lancer le jeu en mode console**, exécutez le fichier `demineur_console.py`.
- La taille du tableau par défaut est de 5 rangées et 5 colonnes, avec 5 mines cachées. Nous vous conseillons de commencer par programmer votre jeu avec ces valeurs par défaut, pour réduire la complexité. Quand tout fonctionnera bien, vous pourrez ensuite tester votre programme avec de plus grandes valeurs.
- Dans le tableau, les cases vides sont 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.
- Le programme devrait vérifier toutes les erreurs d'entrées de l'utilisateur. Il y a trois types d'erreur :

  - le joueur entre des coordonnées qui ne sont pas des entiers pour les numéros de rangée et de colonne ;
  - le joueur choisit des coordonnées qui ne sont pas dans le tableau ;
  - le joueur entre des coordonnées pour une case déjà dévoilée.

  En cas d'entrée erronée, le programme demande à l'utilisateur d'entrer de nouvelles coordonnées.

- 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.

Important : Nous mettons à votre disposition trois exemples de résultats de l'exécution du programme final en mode console, dans des fichiers texte.

5.1 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 la classe `Partie` permettent de demander à l'utilisateur les coordonnées de la case à dévoiler, puis appellent les méthodes de la classe `Tableau` pour valider les coordonnées, dévoiler la case et afficher le tableau. À chaque dévoilement, on 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.

5.2 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 paires (x, y) représentant les coordonnées d'une case (une paire est en fait un tuple de deux éléments). 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 valeurs 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. Notez que la majorité des méthodes de la classe `Tableau` ont des tests unitaires qui y sont associés, que vous pouvez exécuter pour valider vos réponses en exécutant directement le fichier `tableau.py`.

5.3 La classe `Case`

La classe `Case` modélise les données et les traitements d'une case d'un tableau du jeu démineur. Nous vous fournissons le code de cette classe ; vous n'avez pas à modifier ce fichier.

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.

5.4 Algorithme de haut-niveau

1. Dans le fichier `demineur_console.py` (que vous n'avez pas à modifier), on crée un objet de la classe `Partie`, puis on appelle la méthode `jouer`. Celle-ci doit d'abord demander à l'utilisateur les caractéristiques du tableau de jeu : le nombre de lignes, le nombre de colonnes, et le nombre de mines cachées. On initialise ensuite un objet de la classe `Tableau`, qui est référé par l'attribut `tableau_mines`. Lors de l'instanciation de `tableau_mines`, un dictionnaire (dont les clés sont les coordonnées des cases et les clés sont les cases elles-mêmes) est initialisé.

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 obtiendrez quelque chose qui ressemble à ça si vous l'affichez avec la méthode `afficher_solution()` :

```
  | 1 2 3 4 5
--+-----------
1 | M 1 1 M 1
2 | 1 2 2 2 1
3 | 0 1 M 2 1
4 | 1 2 2 2 M
5 | 1 M 1 1 1
```

3. Affichez le tableau pour le joueur où les cases qui n'ont pas été dévoilées apparaissent comme étant cachées à l'aide de la méthode `afficher_tableau()`. 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 (ceci est bien illustré par les exemples d'exécutions du programme qui vous sont fournis). Pour programmer la méthode `afficher_tableau()`, il est conseillé de s'inspirer fortement de la méthode `afficher_solution()` fournie.

4. Demandez au joueur d'entrer les coordonnées d'une case (numéros de rangée et de colonne) du tableau qu'il veut dévoiler. Les rangées et les colonnes sont numérotées à partir de 1.

5. Si le joueur dévoile une mine, montrez la solution et informez-le de sa défaite. 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.

5.5 Effet de dévoilement en cascade

Si le joueur choisit une case vide dans le tableau, dévoilez les cases des voisins immédiats. Si une de ces cases voisines est aussi vide, alors ses voisins à elle devraient 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 se programme naturellement avec l'appel d'une fonction récursive.

Pour les cas de base (quand la fonction n'effectue pas d'appels récursifs), vous devez considérez ces situations :

- la case n'est pas vide (attention, vous devez non seulement arrêter le dévoilement en cascade lorsque vous rencontrez une case voisine d'une mine, mais aussi vous assurez qu'aucun dévoilement en cascade n'est entamé si on clique sur une case minée) ;
- tous les voisins de la case sont déjà dévoilés (sans cela, vous risquez fortement d'avoir une récursion infinie).

Autrement, le dévoilement en cascade devrait effectuer un ou plusieurs appels récursifs. Normalement, un appel sur une case en particulier devrait lancer autant d'appels récursifs que la case a de voisins non dévoilés. Vous pouvez placer votre appel récursif dans une boucle *for* qui itère sur les voisins de la case en cours, par exemple.

Pour visualiser et tester votre dévoilement en cascade, il est suggéré de générer une grande carte avec peu de mines.

6 Spécifications du jeu en mode graphique (GUI)

Bonne nouvelle : Le gros du travail pour l'interface graphique a déjà été fait pour vous ! Une fois que votre jeu fonctionne correctement en mode console, vous n'aurez que **quatre méthodes** à compléter dans la classe `InterfacePartie` :

- `nouvelle_partie()`
- `configurer_preset()`
- `detecter_fin()`
- `quitter()`

Pour lancer le jeu en mode graphique, exécutez le fichier `demineur_graphique.py`.

Conseil : Prenez le temps de lire sommairement le code fourni dans `interface_partie.py` afin de comprendre quelles méthodes sont à votre disposition et comment elles interagissent. Vous pourrez ainsi identifier quelles méthodes de la classe `Tableau` (que vous avez implémentées pour le mode console) vous pouvez réutiliser.

6.1 Déroulement du jeu

Le jeu se déroule comme pour le mode console : le joueur dévoile des cases en essayant d'éviter les mines. S'il dévoile une mine, la partie est perdue et un message s'affiche pour informer le joueur de sa défaite. S'il dévoile toutes les cases sans mine, la partie est gagnée et un message s'affiche pour informer le joueur de sa victoire. Peu importe l'issue de la partie, toutes les cases sont dévoilées. Contrairement au mode console, terminer une partie ne doit pas fermer le jeu. L'utilisateur peut démarrer une nouvelle partie ou quitter en cliquant sur l'élément de menu associé.

6.2 Connecter l'interface à la classe Tableau

Il ne faut pas reprogrammer les fonctionnalités du jeu dans le fichier de l'interface ! Il faut plutôt réutiliser le code que vous avez déjà. Par exemple, le dévoilement d'une case devrait utiliser la méthode `devoilement_en_cascade()` de la classe `Tableau`. D'autres méthodes de la classe `Tableau` vous seront utiles pour déterminer si la partie est terminée.

6.3 Fonctionnalités de l'interface

En plus de pouvoir jouer au démineur, votre interface doit posséder au minimum les fonctionnalités suivantes.

- Un élément de menu « Nouvelle partie » permet de redémarrer la partie en tout temps. La configuration (taille du tableau et nombre de mines) est la même que pour la partie en cours.
- Un élément de menu « Quitter » permet de quitter le programme en tout temps. Avant de quitter, une fenêtre de dialogue doit demander la confirmation de l'utilisateur.
- Un élément de menu « Personnaliser... » ouvre une nouvelle fenêtre, qui doit offrir un moyen de changer la configuration du tableau (le nombre de lignes, de colonnes et de mines). La validation des entrées (par exemple, si l'utilisateur entre *trois* au lieu de 3), n'est pas obligatoire, mais il s'agit d'une fonctionnalité facultative.
- Votre interface doit permettre de sauvegarder l'état de la partie dans un fichier texte et de continuer une partie en chargeant une partie à partir d'un fichier texte. La sauvegarde et le chargement se font à l'aide d'un fichier nommé `sauvegarde.txt`. Cette fonctionnalité a déjà été implémentée pour vous. Autrement dit, vous n'avez rien à faire ici.
- N'oubliez pas de détecter la victoire ou la défaite, d'afficher un message en conséquence et de dévoiler toutes les cases.

6.4 Exemple d'exécution

Un exemple d'exécution des fonctionnalités de l'interface graphique sera présenté en classe.

7 Comment attaquer le problème

À première vue, le problème à résoudre peut paraître intimidant. 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 autres méthodes ».

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.

Cela peut vous sembler très gros, mais rappelez-vous

- que les règles du jeu finiront par vous venir intuitivement ;
- qu'une bonne partie du code est déjà programmée pour vous ;
- que les méthodes que vous devez implémenter sont bien décrites dans leur documentation ;
- que vous pouvez poser des questions sur le forum ;
- et que vous pouvez partager la charge de travail avec votre coéquipier(e)!

8 Travail à faire

Vous devez programmer toutes les méthodes qui ne sont pas déjà programmées, ainsi que compléter celles qui sont incomplètes. Chacune de ces méthodes possède un ou plusieurs commentaires `# TODO`, qui vous indiquent que vous devez programmer la méthode, ou la compléter. Attention de ne pas en oublier !

Si vous comprenez bien l'interaction entre les diverses méthodes, chacune d'entre elles devrait être utilisée à au moins un endroit dans votre programme. Notez que pour les méthodes de la classe `Tableau`, à l'exception du dévoilement en cascade, des tests unitaires vous sont fournis, ce qui vous permettra d'avancer avec l'ordre inverse tout en validant la plupart de vos réponses.

Il vous sera **permis** de définir d'autres méthodes que celles qui vous sont fournies (à condition de bien les documenter). Mais **attention**, vous devez vous assurer que toutes les méthodes qu'on vous demande d'implémenter le soient. Dans l'exemple suivant, la version de gauche est correcte mais pas celle de droite :

```python
def nouvelle_methode(self, x):
    """
    Docstring dans le style Google
    """
    # traitement

def methode_demandee(self, x):
    return self.nouvelle_methode(x)
```

```python
def nouvelle_methode(self, x):
    # traitement

def methode_demandee(self, x):
    pass # ne fait rien puisque
    # le code utile est dans
    # l'autre méthode
```

En d'autres termes, vous pouvez diviser les méthodes qui vous sont demandées en sous-méthodes, mais vous ne devez pas les remplacer. Vous ne devez pas créer de nouveaux fichiers ou de nouvelles classes.

9 Résumé des fichiers à exécuter

Mode	Fichier à exécuter	Description
Console	`demineur_console.py`	Version texte du jeu
Graphique (GUI)	`demineur_graphique.py`	Version avec interface Tkinter

10 Exemple d'exécution

Trois fichiers texte vous sont fournis. Ils contiennent l'affichage en console généré par une solution du travail.

- `exemple_victoire.txt` : une partie complète où le joueur remporte en ne cliquant sur aucune mine.
- `exemple_defaite.txt` : une partie complète où le joueur échoue en cliquant sur une mine.
- `exemple_recursivite.txt` : une partie partielle où le joueur déclenche un grand dévoilement en cascade.

Des exemples d'exécution du jeu en mode graphique vous sont également fournis.

11 Ce que vous devez remettre

Votre programme doit être rédigé à même les 3 fichiers Python `tableau.py`, `partie.py` et `interface_partie.py`, que vous devez compresser dans une archive Zip avec le fichier **correction.txt**. Vous n'avez pas à inclure les autres fichiers.

Assurez-vous que vous remettez tous les fichiers nécessaires à l'exécution de votre TP. Nous vous recommandons fortement de créer un **dossier** dans lequel vous mettrez les fichiers relatifs à votre TP, et **rien d'autre**. Nous ne pourrons pas donner de points à votre travail si vous remettez le mauvais fichier. Les **capsules** *Les fichiers sous Windows* et *Les fichiers sous Mac OS* devraient vous aider à bien vous organiser.

Aussi 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. Il en sera de même pour tous les étudiants qui ne respecteront pas l'interface des méthodes ou des classes.

12 Modalités d'évaluation

Ce travail sera évalué sur 100 points, et la note sera ramenée sur 10. Voici la grille de correction :

Élément	Pondération
Le programme initialise correctement le tableau de mines	15 points
Le programme affiche le tableau selon l'état de dévoilement des cases	5 points
Le programme valide les coordonnées	5 points
Le programme valide les entrées de l'utilisateur	5 points
Le programme permet de choisir les options de jeu	5 points
Le programme implémente l'effet de dévoilement en cascade récursif	20 points
Le programme permet un bon déroulement de la partie	10 points
La partie se termine si le joueur déclenche une mine	5 points
La partie se termine si le joueur a dévoilé toutes les cases sans mine	5 points
Interface graphique fonctionnelle	10 points
Respect de la décomposition fonctionnelle demandée	10 points
Qualité du code (noms de variables, style, commentaires, documentation)	2 points
Utilisation de Git	3 points

Utilisation de Git

Vous devrez compléter la partie intitulée *« Lien vers le dépôt GitHub PRIVÉ de votre TP : »* dans le fichier `correction.txt` fourni avec cet énoncé.

De plus, jusqu'à 10 points de pénalité pourront être enlevés pour la mauvaise gestion de compte GitHub, incluant une pénalité individuelle à un membre de l'équipe qui n'aurait fait aucun *commit* au dépôt GitHub.

13 Remarques additionnelles

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 : Les travaux pratiques doivent être impérativement remis via le portail de cours. Aucune remise par courriel n'est acceptée. Un travail qui ne respecte pas les directives de

remise se verra pénalisé de 20%. Tout travail remis en retard se verra pénalisé de 25% par jour de retard. Chaque journée de retard débute dès la limite de remise dépassée (dès la première minute). Un retard excédant 2 jours provoquera le rejet du travail pour la correction et la note de 0 pour ce travail.

Remises multiples : Il vous est possible de remettre votre TP plusieurs fois sur le portail des cours. La dernière version sera considérée pour la correction.

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 indentation, marge à droite, et souligné).

Documentation du programme : Si vous créez de nouvelles méthodes, vous devez les documenter à l'aide de « docstrings ». La seule exception à cette règle sont les fonctions de tests unitaires, pour lesquelles nous n'exigeons pas de commentaires.

Bon travail !