Projet en python : le démineur

Le principe du jeu

Le projet consiste à implanter en `Python` un jeu de *démineur* (« *minesweeper* »).

régles classiques

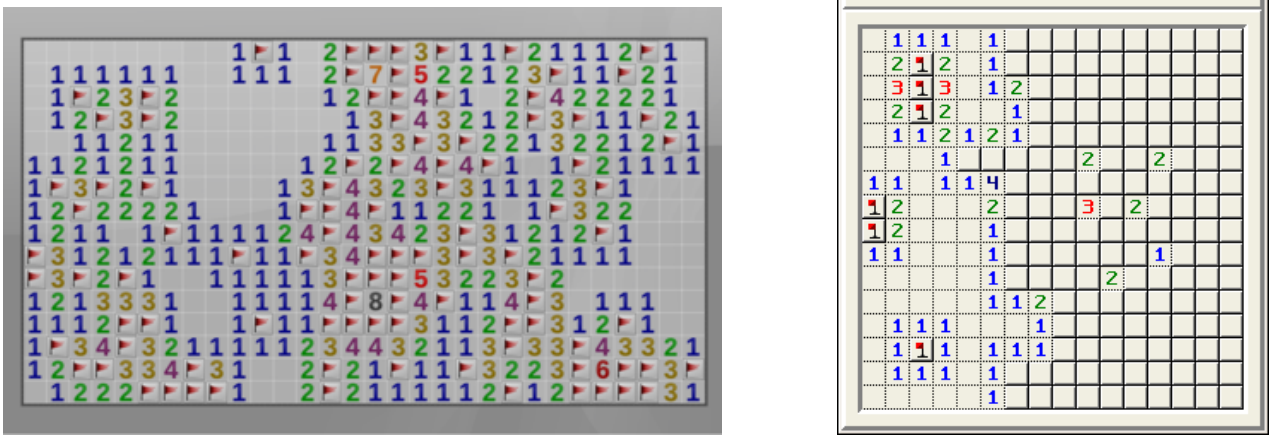
Le démineur est un jeu de réflexion dont le but est de localiser des mines cachées dans une grille représentant un champ de mines virtuel, avec pour seule indication le nombre de mines dans les zones adjacentes.

Le champ de mines du démineur est représenté par une grille, qui peut avoir différentes formes : deux ou trois dimensions, pavage rectangulaire ou non, etc.

Chaque case de la grille peut soit cacher une mine, soit être vide. Le but du jeu est de découvrir toutes les cases libres sans faire exploser les mines, c'est-à-dire sans cliquer sur les cases qui les dissimulent.

Lorsque le joueur clique sur une case libre comportant au moins une mine dans l'une de ses cases avoisinantes, un chiffre apparaît, indiquant ce nombre de mines. Si en revanche toutes les cases adjacentes sont vides, une case vide est affichée et la même opération est répétée sur ces cases, et ce jusqu'à ce que la zone vide soit entièrement délimitée par des chiffres. En comparant les différentes informations récoltées, le joueur peut ainsi progresser dans le déminage du terrain. S'il se trompe et clique sur une mine, il a perdu.

On peut signaler les cases contenant des mines présumées par un drapeau en cliquant sur le bouton droit de la souris — mais ce n'est aucunement obligatoire. Il faut faire attention à ne pas signaler une case saine par un drapeau, car cela peut induire en erreur ; ce n'est toutefois pas aussi pénalisant que de découvrir une mine.



(a) Version `Kmines` (`Linux`). (b) Version `Mines-Perfect` (`windows`).

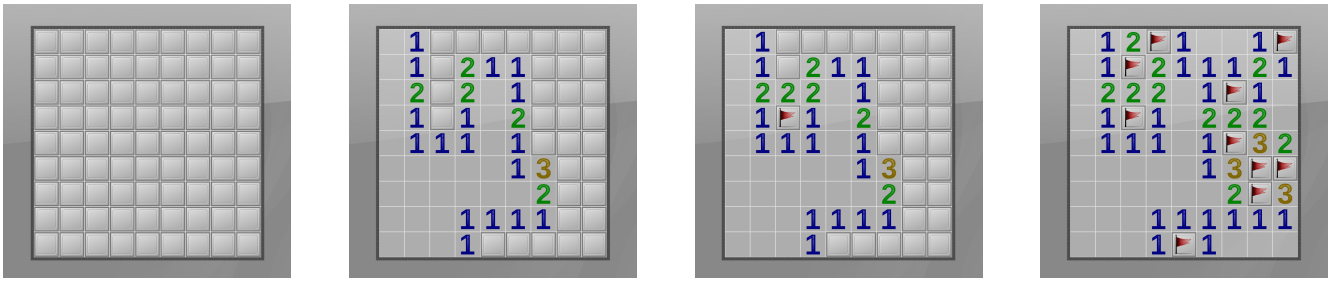
Figure 1 – Différentes versions classiques du jeu de démineur.

Déroulement d'une partie

Le but du jeu est de marquer toutes les bombes de la grille sans en découvrir une seule (creuser une case contenant une bombe la fait exploser).

La partie se termine lorsque :

- le joueur a découvert une case contenant une bombe : dans ce cas, celle-ci explose et la partie est perdue
- le joueur a découvert toutes les cases ne contenant pas de bombe : dans ce cas, le terrain est complètement déminé et la partie est gagnée



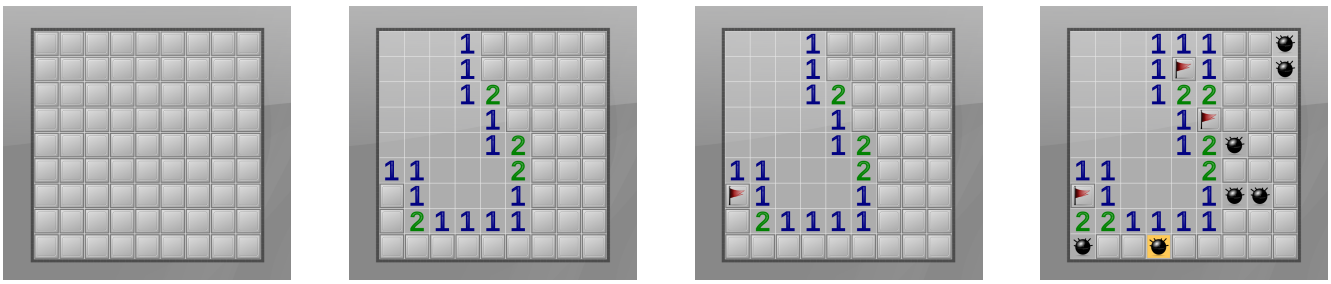
(a) Grille initiale.

(b) Après avoir creusé dans la case du coin en haut à gauche.

(c) Marquage d'une bombe.

(d) Victoire.

FIGURE 2 – Exemple de partie gagnée.



(a) Grille initiale.

(b) Après avoir creusé dans la case du coin en haut à gauche.

(c) Marquage d'une bombe.

(d) Défaite après un mauvais choix.

FIGURE 3 – Exemple de partie perdue.

Travail demandé

Vous devez, en respectant les consignes, en utilisant les formats et en implantant les fonctions imposés par les instructions ci-dessous, réaliser un programme `Python` qui permet de jouer une ou plusieurs parties de démineur à un joueur selon les règles définies ci-dessus. Après chaque partie, il sera proposé à l'utilisateur de rejouer.

Instruction 0. La grille du jeu devra nécessairement être implantée par une structure `Python` de type *liste de listes*. Toute librairie ou fonction non basique `Python` sur les listes simples ou multidimensionnelles sont interdites.

Instruction 1. Implanter, dans un fichier `affichage.py` la procédure `afficher_grille` permettant d'afficher une grille de jeu. paramètre `reveler_bombes` permettant d'afficher complètement la grille en fin de partie (de valeur par défaut `False`)

Instruction 2. Implanter, dans un fichier `saisie.py`, le code des fonctions permettant toutes les saisies clavier nécessaires au déroulement du jeu (saisie des touches de jeu, saisie du choix du joueur pour rejouer ou sauvegarder une partie). La saisie au clavier devra être *protégée* : le programme ne devra pas en aucun cas s'interrompre si le joueur ne saisit pas les touches prédéfinies et nécessaires à la jouabilité du jeu.

Instruction 3. Implanter dans un fichier `initialisation.py`

- `generer_bombes` de paramètres deux entiers naturels `n` et `nb_bombes` et retournant une liste de coordonnées de `nb_bombes` bombes placées aléatoirement dans une grille de taille $n \times n$
- `saisir_bombes`
- `generer_grilles`
-

Instruction 4. Implanter dans un fichier `noyau.py` les fonctions

- `creuser`
- `compter_case`, de paramètre une grille et un état parmi la liste d'états possibles d'une case
- `voisinage`

- `incrementer_voisins`
- `decouvrir` de paramètres la grille, et deux entiers correspondant aux coordonnées de la case à creuser
  retourne un booléen : faux si la case est une bombe, et vrai sinon Dans le cas où la case n'est pas une bombe, cette fonction doit marquer toutes les cases « sûres » en état `verifie`, suivant la règle usuelle du démineur
  c'est la fonction principale du jeu du démineur : elle permet de découvrir la plus large zone sécurisée, c'est-à-dire ne contenant pas de bombes, et dont les voisines ne contiennent pas non plus de bombes

Instruction 5. Implanter, dans un fichier `saisie.py`, le code des fonctions permettant de

Instruction 6. Créer le fichier principal nommé `demineur.py`. C'est dans ce fichier que seront importés les autres fichiers `Python` à l'aide d'instructions du type :
`from nom_module import *` (sans l'extension `.py`)
et que la boucle principale de jeu sera appelée et exécutée, dans la fonction `jouer_au_demineur`.
procédure `accueil`

Instruction 7. Implanter, dans un fichier `sauvegarde.py`,
`sauvegarder_grille` et `restaurer_grille`
Implanter dans ce même fichier, à partir d'une liste de dimension 3, des fonctions permettant de construire l'historique de la partie, et de le sauvegarder dans un fichier externe `historique.txt`.

Instruction 8. `time` pour mesurer la durée (en secondes) de la partie

Instruction 9. `classement`

De plus, les contraintes suivantes devront impérativement être respectées.

Contrainte sur les structures de données. une grille est une liste de dimensions 2, *i.e.* une liste de listes

une case est un dictionnaire, de clés `'etat'`, `'bombe'`, `'voisines_bombes'`, `'x'` et `'y'` `'etat'` : `non verifie`, `verifie`, `marque bombe` : valeur booléenne `voisines_bombes` : valeur entière (nombre de bombes voisines, ou -1 si la case est une bombe `'x'` et `'y'` : valeurs entières (coordonnées de la case)

Ainsi, si la case située dans le coin en haut à gauche de la grille contient une bombe, elle sera représentée par le dictionnaire `{'etat': 'non verifie', 'bombe': True, 'voisines_bombes': -1, 'x': 0, 'y': 0}`

De même, le dictionnaire `{'etat': 'verifie', 'bombe': False, 'voisines_bombes': 3, 'x': 2, 'y': 4}` représentera la case située en colonne E (quatrième colonne, en comptant à partir de 0) et en ligne 3 (donc ligne numéro 2, en comptant à partir de 0), qui a déjà été découverte et ne contient pas de bombes, mais dont 3 cases voisines sont des bombes.

Contraintes sur la structures des fichiers. Le programme devra être découpé en quatre fichiers `Python`, chacun ayant un rôle bien défini et regroupant les fonctions correspondantes :

- `affichage.py` : module de gestion des sorties (affichage) du programme ; il contient la (ou les si nécessaires) procédure(s) d'affichage
- `saisie.py` : module de gestion des entrées (saisie au clavier) ; il contient les fonctions de saisie (saisie qui doit être protégée, *c.f.* ci-dessous)
- `noyau.py` : module contenant toutes les fonctions et procédures nécessaires (excepté celles des entrées/-sorties) au déroulement d'une partie
- `demineur.py` : fichier principal, à exécuter pour jouer une partie ; les autres modules y sont importés, et la boucle principale de jeu est ensuite implantée
- `parametres.py` : fichier

L'import de la totalité des fonctions dans un autre module se fait par l'instruction :
`from module_a_importer import *`

Contrainte sur la saisie. Il est de plus demandé de gérer la *protection de la saisie au clavier* : le programme ne devra pas s'interrompre si le joueur ne saisit pas les bons types de données demandés. Par exemple, si on saisit « `!!?3@nimp&#%` » comme numéro de dé, le programme ne devra pas s'arrêter avec un message d'erreur mais demander de ressaisir un numéro de dé.

Contrainte de paramétrabilité. fichier `parametres.py` paramètres obligatories :

- `TAILLE_GRILLE`
- `NB_BOMBES`
- `DIFFICULTE`
- `RATIO_BOMBES_MAX`
- `PLACEMENT_MANUEL`
- `SYMBOLE_BOMBE`
- `SYMBOLE_DRAPEAU`

Consignes et évaluation

Le code source du projet doit être envoyé par mail à l'adresse `senotprof@gmail.com` **avant le jeudi 17/10/2024 à 23h59**. Le sujet du mail doit être intitulé exactement au format suivant : `[NSI] Prénom1 Prénom2 - projet yahtzee`. Le code source devra être nommé au format `prenom1_prenom2_yahtzee` (ni majuscule, ni accent, ni espace) et joint au mail dans un format de dossier compressé de type `zip` (ou d'archive compressée au format `.tar.gz`), aucun format propriétaire, type `.rar`, ne sera accepté. Aucun délai supplémentaire ne sera accordé, la date butoir est immuable, et tout retard sera pénalisé : il convient donc de commencer le projet au plus tôt et de bien organiser son planning. Le moindre non-respect de ces consignes aura pour conséquence une non-correction du projet, qui sera alors évalué par la note minimale.

Le projet est **à coder obligatoirement en binôme** (sauf dérogation pour un unique trinôme, si l'effectif de classe est impair). La note sera commune au binôme, sauf cas de travail déséquilibré flagrant.

Il n'est pas interdit (et c'est même conseillé) de discuter du projet entre les différents groupes et de s'échanger des idées ou de demander un peu d'aide (à commencer par celle de votre enseignant), mais l'intégralité du code devra être produite par les membres du groupe (une fonction ou une portion de code bêtement recopiée ne sera pas utile dans un autre contexte que le programme initial, et surtout ne sera ni comprise, ni assimilée, et constituera, en plus des risques de sanctions pour fraude et plagiat, une perte de temps pour tout le monde, élèves plagieurs, élèves plagiés et enseignants). Un entretien individuel avec chaque binôme, sous forme d'une mini-séance de questions spécifiques, pourra se tenir après la remise des projets afin de s'assurer que le programme a bien été conçu et implanté en `Python` par les membres du groupe.

Tout plagiat fera l'objet de sanctions avec une tolérance strictement négative.

L'évaluation prendra en compte :

- les erreurs de syntaxe : les fichiers `.py` doivent être directement **interprétés sans erreur** par la console `Python`
- l'**originalité du code** (dans le sens de « code personnel et individuel »)
- l'implantation des fonctions imposées et le **respect des consignes** : règles du jeu, cahier des charges, consignes sur le code, les structures de données et les spécifications des fonctions, mais aussi les consignes annexes, comme l'intitulé du mail et les formats d'envoi, ainsi que bien évidemment, le respect des délais
- la **propreté et la lisibilité du code** ainsi que tout ce qui facilitera sa compréhension par le correcteur : docstrings et commentaires, noms des variables, docstrings et commentaires, noms des fonctions, docstrings et commentaires, aération et organisation du code, docstrings et commentaires, *etc*
- la **correction** et la performance du code
- la **facilité d'utilisation** et la jouabilité
- la **paramétrabilité** et possibilité de généralisation du jeu (choix des symboles de jeu, taille de la grille, *etc*)

Suivi et rapport de projet

Le suivi de l'évolution et la finalisation du projet sera constituée de :

- un bilan intermédiaire, dont la date sera précisée ultérieurement, et qui consistera en la rédaction d'un mail détaillant l'avancement du projet, ainsi que les fichiers `Python` en l'état à la date du bilan intermédiaire
- un rapport écrit, de préférence un tapuscrit, d'entre 3 et 5 pages contenant :
  - la répartition détaillée du travail dans le groupe
  - la progression chronologique et les dates approximatives des différentes versions
  - un schéma simple présentant les interfaces et interactions principales des différents modules et fonctions

  - le récit détaillé d'un problème spécifique rencontré lors du déroulement du projet (chronologie, nature et contexte du problème) et de sa gestion et résolution (choix faits et justifications techniques de la solution implantée)
- d'une mini-séance d'au maximum 10 min consistant en quelques questions ciblées à chacun des membres du groupe (il ne s'agit en aucun cas d'une présentation devant toute la classe)

Travail optionnel proposé (c'est quasiment du bénévolat : ça rapporte que dalle en points)

Si le temps le permet, il est possible, pour se familiariser avec `Python` et la conduite d'un projet, d'apporter diverses améliorations au programme, comme par exemple :

- suivre l'adage des programmeurs : « *less is more* », et simplifier au maximum le code
- améliorer la présentation : menu simple, grille en couleur
- développer une interface graphique (par exemple avec le module `pygame`)
- créer un menu présentant les options : par exemple proposer de rejouer en fin de partie, compter les parties gagnées par les joueurs, identifier les joueurs par leurs prénoms, *etc*
- programmer l'ordinateur pour qu'il joue une partie de démineur de manière autonome suivant une stratégie prédéfinie

L'éventuelle implantation d'un bonus ne sera prise en compte que si tout le reste est fonctionnel et conforme au cahier des charges (et même dans ce cas, une fraction de point bonus n'est même pas sûre d'être attribuée).

Annexe : un exemple

Un exemple d'interface et de déroulement d'un début de partie est donné ci-dessous. Cet exemple a pour vocation à donner des idées et servir d'inspiration, et non à être recopié à l'identique.

```
    *** BIENVENUE AU DEMINEUR ***

    La partie est gagnée lorsque toutes les cases ne contenant pas de bombes ont été découvertes.
    Elle est perdue lorsqu'une bombe explose, c'est-à-dire lorsqu'on creuse une case contenant une bombe

    Les coordonnées de la case devront être saisies au format ColonneLigne, éventuellement suivies d'un point d'exclamation pour marqu
    Exemples :
        - B3 pour découvrir la case en colonne B et en ligne 3
        - A1! pour marquer la case en colonne A et en ligne 1



   A  B  C  D  E  F  G
1
2
3
4
5
6
7

Saisir les coordonnées de la case à creuser : A0
Saisir les coordonnées de la case à creuser : A1

   A  B  C  D  E  F  G
1  .  1
2  .  1
3  .  1
4  1  2
5
6
7

Saisir les coordonnées de la case à creuser : A8
Saisir les coordonnées de la case à creuser : A6

   A  B  C  D  E  F  G
1  .  1
2  .  1
3  .  1
4  1  2
5
6
7

BOUM ! ben c'est perdu donc.

   A  B  C  D  E  F  G
1  .  1  1  1  .  .  .
2  .  1  X  1  .  .  .
3  .  1  2  2  1  .  .
4  1  2  2  X  1  1  1
5  X  4  X  2  1  2  X
6  X  X  3  3  1  3  X
7  2  3  X  2  X  2  1
```