

Projet ISN - Terminale S - Démineur

BOURGET Alexis, LE TERTE Dorian, MIGADEL Kevin - TS1

Année Scolaire 2015-2016



Table des matières

1	Présentation du projet	1
2	Les idées et objectifs du projet	2
2.1	Cahier des charges du projet	2
2.2	Moyens de réalisation du projet	2
3	Organisation du travail en groupe	3
3.1	Répartition des tâches	3
3.2	Moyens de collaboration	3
4	Réalisation	4
4.1	Génération du terrain	4
4.2	Interface du démineur	5
5	Bilan du projet	7
5.1	Ce qu'il m'a apporté	7
5.2	Ce que nous pourrions améliorer	7

Listings

1	Exemple de documentation	8
2	Exemple des tests réalisés (ici le fichier <i>terrain.py</i>)	8
3	Premier problème d'index	8
4	Second problème d'index	8
5	Fonction <i>entourage_case</i> finale	9
6	Fonction <i>canvas_plateau</i>	10
7	Version 1 de <i>cases_img</i>	11
8	Version finale de <i>cases_img</i>	11
9	Fonction <i>maj_revele_case</i>	12

1 Présentation du projet

Pourquoi ce projet ? Nous avons choisi de créer un démineur car, malgré son apparente simplicité, ce n'est pas un jeu simple. En effet, il faut créer un générateur de terrain qui soit le plus aléatoire et le plus rapide possible et qui place correctement les numéros sur les cases. De plus, l'interface graphique associée doit répondre à plusieurs événements (clic sur le terrain ou non, clic gauche ou droit, clic sur une case découverte ou non, ...).

Ce qui existait déjà : Nous avons effectué des recherches pour nous informer sur les solutions existantes sur le Web (sur Github notamment). Nous avons trouvé au final assez peu de ressources : nous cherchions principalement des sources graphiques pour les éléments du jeu mais n'en n'avons pas trouvées et les avons faites à la main.

Pour ce qui est du code, nous avons évité de regarder ce qui avait déjà été fait pour ne pas copier, surtout quand il s'agissait de code Python.

Ce que nous avons obtenu des recherches sur le Web : J'ai lu beaucoup de code sur Github, que ce soit en C, C++, C# ou Java sur Github afin d'avoir une idée de la logique derrière un démineur. Ce qui est ressorti est qu'il existe tellement d'approches différentes qu'il y en a quasiment une par projet.

Les codes lus étaient soit en anglais soit en français. J'ai trouvé des exemples basés sur la programmation orienté objet, d'autres sur un paradigme fonctionnel et enfin certains utilisaient des boucles événementielles.

Principe : Notre démineur est classique : il faut révéler toutes les cases non-minées du terrain sans toucher une case minée au passage. Pour cela, les cases sont numérotées en fonction du nombre de mines dans leur entourage direct. Le joueur peut placer/supprimer des drapeaux à sa guise.

2 Les idées et objectifs du projet

2.1 Cahier des charges du projet

Pour réaliser ce projet au mieux possible avec nos capacités nous avons choisi de limiter la charge de travail. Ainsi ils existent de nombreuses améliorations possibles pour notre démineur. Voici toutefois les idées qui nous ont servies de base de départ :

- Génération d'un terrain miné aléatoire, de la taille demandée (avec un taille minimum et maximum),
- Adapter la taille des cases aux dimensions du terrains,
- Pouvoir relancer une partie sans relancer l'application,
- Pouvoir placer/enlever des drapeaux sur les cases non-découvertes,
- Avoir un chronomètre mesurant la durée de la partie,
- Placer une mine de couleur différente des autres pour signaler celle qui, le cas échéant, a fait perdre la partie,
- Afficher un message de fin de partie,
- Pouvoir propager la révélation des cases si l'on clique sur une case sans mines aux alentours ou pour révéler le terrain en fin de partie

2.2 Moyens de réalisation du projet

Choix du langage : Nous avons choisi d'utiliser le langage *Python* en version 3.5. Ce choix a été motivé par plusieurs raisons. La première raison était que nous avons tous étudié ce langage au cours de l'année et le connaissions donc un minimum. La seconde raison est la communauté Python, qu'elle soit française ou anglaise, qui est très développée et active et donc plus à même de nous aider en cas de problème important. La dernière raison était plus spécifique à notre groupe, puisqu'elle vient du fait que j'ai déjà beaucoup codé en Python, que ce soit pour des programmes graphiques ou en ligne de commande.

Nous avons utilisé uniquement la bibliothèque *tkinter* pour notre projet, afin de le rendre jouable sur un maximum de systèmes d'exploitations d'un coup.

Choix du 'type' de code : Nous avons choisi d'écrire du code sous forme de fonctions et en partant sur une gestion événementielle du démineur. Cela nous a évité de manipuler les classes, trop compliquées pour nous. Nous avons toutefois fait une exception pour le chronomètre car la solution utilisant une classe était à la fois la plus simple et la plus élégante.

3 Organisation du travail en groupe

3.1 Répartition des tâches

Nous avons réparti les différentes tâches au sein du groupe de la manière suivante, en tenant compte de nos niveaux respectifs et de nos envies.

Par exemple, j'avais envie de m'occuper de la génération du terrain car cela me semblait un défi intéressant, je l'ai donc choisi quand personne n'a protesté contre.

- Alexis : *Génération du terrain + interface*,
- Dorian : *Gestion du temps*,
- Kévin : *Gestion des actions pour rejouer et pour perdre*.

3.2 Moyens de collaboration

Skype : Nous avons utilisé Skype pour nous parler régulièrement et échanger nos progrès sur le code du démineur. Nous avons organisé plusieurs soirées/après-midi où nous étions deux ou trois à discuter sur comment coder une idée particulière afin qu'elle s'intègre dans le projet et s'assurer qu'elle fonctionne ailleurs que sur l'ordinateur du codeur originel.

Github : Nous avons aussi utilisé Github car j'en avais déjà l'usage pour d'autres projets et que son utilisation basique est assez simple. Nous avons utilisé deux branches : une branche *dev* pour nous échanger les fichiers sans qu'ils soient forcément compatibles et la branche de base *master* pour le programme fonctionnel. Cela nous assurait de plus une sauvegarde sûre des fichiers de code et des éléments graphiques.

Documentation : J'ai beaucoup insisté pour que la documentation et les commentaires soient aussi complets et détaillés que possibles. Cela nous a permis de partir du travail d'un autre sans avoir à recomprendre sa manière de réfléchir en lisant du code et nous rendait capable de travailler de notre côté sans avoir besoin d'aide à chaque utilisation d'un code écrit par un autre membre du projet.

Ainsi les signatures des fonctions devaient être aussi complètes que possible, avec la valeur de retour signalée (grâce aux possibilités de Python 3.5).

Documentation : voir Annexes - Listing 1

4 Réalisation

Dans cette partie, je vais présenter ce que j'ai fait et les difficultés rencontrées durant la réalisation des tâches qui m'étaient attribuées.

4.1 Génération du terrain

Objectif : Je devais créer un ensemble de fonctions capables de faire les actions suivantes :

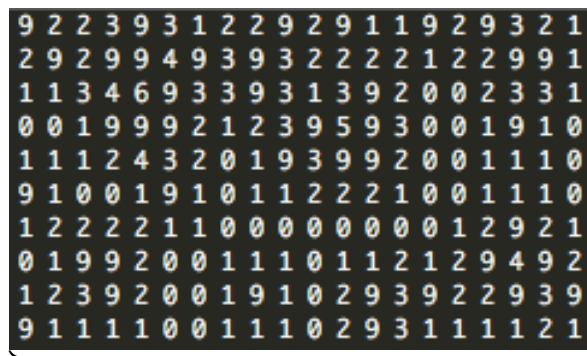
- Générer un terrain de largeur et la hauteur demandées,
- Placer les mines sur ce terrain,
- Placer les chiffres dans les cases entourant les mines.

Réalisation : J'ai donc créé quatre fonctions pour faire cela.

- La première fonction, *entourage_case*, permet de compter les mines autour d'une case.
- *place_mines* place le nombre de mines demandées sur un terrain vide, de façon aléatoire.
- *place_nb_mines* place le nombre de mines adjacentes dans chaque case. Cette fonction utilise la première, présentée plus haut.
- *genere_terrain* fait usage des deux fonctions précédentes pour créer un terrain à partir des choix de l'utilisateur quant à la largeur, la hauteur et le nombre de mines.

J'ai aussi créé une fonction *affiche_terrain* pour pouvoir tester le bon fonctionnement des fonctions précédentes au cours du développement.

Résultats : Voici les résultats que j'obtiens lorsque j'utilise *affiche_terrain* pour générer divers terrains.



9	2	2	3	9	3	1	2	2	9	2	9	1	1	9	2	9	3	2	1
2	9	2	9	9	4	9	3	9	3	2	2	2	2	1	2	2	9	9	1
1	1	3	4	6	9	3	3	9	3	1	3	9	2	0	0	2	3	3	1
0	0	1	9	9	2	1	2	3	9	5	9	3	0	0	1	9	1	0	
1	1	1	2	4	3	2	0	1	9	3	9	9	2	0	0	1	1	1	0
9	1	0	0	1	9	1	0	1	1	2	2	2	1	0	0	1	1	1	0
1	2	2	2	2	1	1	0	0	0	0	0	0	0	0	1	2	9	2	1
0	1	9	9	2	0	0	1	1	1	0	1	1	2	1	2	9	4	9	2
1	2	3	9	2	0	0	1	9	1	0	2	9	3	9	2	2	9	3	9
9	1	1	1	1	0	0	1	1	1	0	2	9	3	1	1	1	1	2	1

Un terrain de 20x10 avec 40 mines.

affiche_terrain : voir Annexes - Listing 2
entourage_case : voir Annexes - Listing 5

Problèmes rencontrés : J’ai eu à corriger des problèmes d’indexation lorsque de je codais la fonction qui compte les mines dans l’entourage d’une case. En effet, avec les cases en bordure il a fallu prendre en compte le fait que pour Python, faire un -1 comme index revient à partir de la fin de liste, ce qui ajoutait parfois une mine qui n’aurait pas due être comptée pour cette case.

Un autre problème a été le placement aléatoires des mines. Avec la première version de ma fonction, les mines étaient toutes alignées sur une colonne à cause d’une erreur de référencement. Après m’être renseigné sur le problème, il a été assez facile à corriger, il venait de la façon dont la double liste représentant le terrain était générée.

Problèmes d’index : voir Annexes - Listings 3 et 4

4.2 Interface du démineur

Objectif : L’interface du démineur devait permettre les actions et montrer les informations suivantes :

- Placer le terrain jouable (pouvoir placer et retirer des drapeaux, révéler les cases sans drapeaux, placer le sprite correspondant à la case),
- Créer les éléments graphiques pour le jeu (puisque nous n’avons rien trouvé à nos goûts),
- Permettre de rejouer une partie,
- Permettre de sélectionner la largeur, la hauteur et le nombre de mines,
- Afficher la durée de la partie.

Réalisation : J’ai utilisé *tkinter* pour réaliser l’interface. Cela permet une portabilité plus importante qu’avec d’autres bibliothèques, facilite l’utilisation du programme et simplifie son partage.

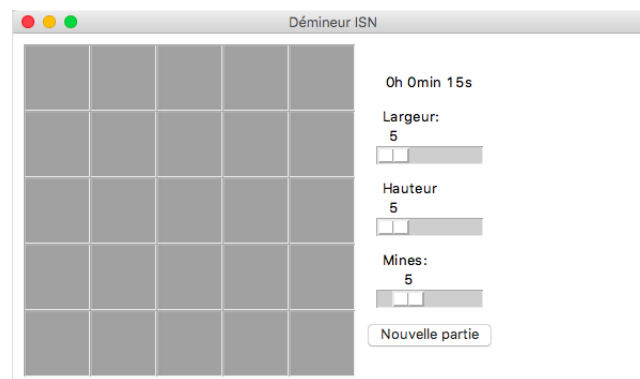
Pour le terrain de jeu, j’ai utilisé un objet *Canvas*. Pour les réglages du jeu, j’ai utilisé des *Scale*, *Label* et *Button* à l’intérieur d’une *Frame*.

Création du plateau de jeu : Le plateau de jeu est créé par la fonction *canvas_plateau(racine : tk.Tk, largeur : int, hauteur : int, col=0, lig=0)*. Elle commence par nettoyer tout ce qui a pu être référencé dans les variables importantes. Ainsi l’ancien plateau est détruit (s’il existe, sinon rien ne se passe) et les variables référençant les cases sont remises à zéro. Les images du jeu sont alors mises à jour à leur tour pour être adaptée à la taille du nouveau plateau, puis le nouveau plateau est initialisé. Enfin les cases sont dessinées une par une pour remplir le plateau et dans le même temps elles

sont référencées de deux manières : par leur position et par leur *ID* de *Label* que *tkinter* leur fournit à leur création.

canvas_plateau : voir Annexes - Listing 6

Résultats : À chaque lancement de l'application, un plateau de 5x5 avec 5 mines est généré, permettant de jouer une partie rapide directement.



Problèmes rencontrés : Le premier gros problème que j'ai rencontré a été un problème avec le *Garbage Collector* de Python, qui supprimait les images quand je tentais de les référencer. J'ai mis du temps à le comprendre car le code que j'avais à l'origine s'exécutait sans problème jusqu'au moment où je cliquais sur une case, moment où la demande d'accès à une image non référencée faisait planter le programme.

Le problème a été réglé assez rapidement une fois que j'ai cherché sur le Web mais le temps que j'ai passé à lire la documentation et le code de *tkinter* n'a absolument pas aidé.

La propagation de la révélation des cases a été le second problème important de l'interface, en présentant deux problèmes en un. Tout d'abord, je n'avais à ce moment aucun moyen d'accéder aux références des cases via leur position, j'ai donc dû me résoudre à créer une variable supplémentaire, *cases_pos*, qui est un dictionnaire de la forme $\{(int, int) : tk.Label\}$ et qui référence les *ID* des cases en fonction de leurs positions en (x, y) . Le second problème était un problème de récursion. En effet, la fonction *maj_revele_case*($x : int, y : int$), dans sa première version, impliquait un grand nombre de boucles inutiles et posait son return trop tard, amenant une erreur de récursion trop importante car elle ne vérifiait pas assez tôt si la case avait été vue auparavant, causant une boucle infinie en repassant sans arrêt sur les mêmes cases.

Problème du Garbage Collector : voir Annexes - Listings 7 et 8
maj_revele_case : Listing 9

5 Bilan du projet

5.1 Ce qu'il m'a apporté

Premièrement, je pense que ce que j'ai le plus appris avec ce projet, c'est à travailler en groupe sur du code. J'utilise souvent Github, mais presque uniquement pour des projets personnels, afin de me donner un moyen de suivi de l'évolution et de faciliter le partage si je veux le passer à quelqu'un.

J'ai aussi pas mal appris sur la bibliothèque standard de Python (je pense notamment à la fonction *after* utilisée par Dorian pour le chronomètre) et sur le module *tkinter*.

Je me suis beaucoup amusé avec la récursion, c'était très intéressant de travailler avec plutôt qu'avec des boucles *while* ou *for*.

Enfin, la gestion de modules s'appelant entre eux, les espaces de noms leur étant attribués et tout le travail pour faire une documentation aussi claire et complète que possible ont été très formateurs dans leurs domaines.

5.2 Ce que nous pourrions améliorer

Les scores Nous avons pensé durant le projet à ajouter des scores avec sauvergarde de ces derniers pour permettre aux joueurs de se comparer à eux mêmes ou entre eux mais nous n'avons pas pris le temps de plus réfléchir à l'idée et ne sommes pas allés plus loin.

Nous avons aussi envisagé d'ajouter du son lors des clics sur le plateau, mais là non plus nous ne sommes pas allés plus loin, principalement par manque de fichiers de son que nous n'avions pas les moyens d'enregistrer correctement.

Annexes : Code

```
1 def genere_terrain(largeur: int, hauteur: int, nb_mines: int) -> (list,
2     tuple):
3     """Génère un terrain comportant nb_mines dans un jeu de largeur *
4     hauteur.
5
6     Arguments:
7     - largeur          - int - largeur en case du terrain,
8     - hauteur          - int - hauteur en case du terrain,
9     - nb_mines         - int - le nombre de mines à placer dans le terrain.
10
11     Retourne:
12     - terrain          - list - le terrain miné avec ses entourages calculés,
13     - pos_mines        - tuple - les positions des mines."""
14
15     # On génère un terrain vide
16     terrain = [[CASE] * largeur for _ in range(hauteur)]
17
18     # On y place les mines
19     terrain = place_mines(terrain, nb_mines)
20
21     # On calcule l'entourage et on récupère le terrain et la position des
22     mines
23     terrain, pos_mines = place_nb_mines(terrain)
24
25     return terrain, pos_mines
```

Listing 1 – Exemple de documentation

```
1 if __name__ == '__main__':
2     affiche_terrain(genere_terrain(20, 10, 40)[0])
3     affiche_terrain(genere_terrain(8, 8, 20)[0])
```

Listing 2 – Exemple des tests réalisés (ici le fichier *terrain.py*)

Problèmes d'index - Fonction *entourage_case* :

```
1 try:
2     if self.terrain[y-1][x-1]:
3         self.terrain[y][x] += 1
4 except IndexError:
5     pass
```

Listing 3 – Premier problème d'index

```
1 try:
2     if self.terrain[abs(y-1)][abs(x-1)]:
3         self.terrain[y][x] += 1
4 except IndexError:
5     pass
```

Listing 4 – Second problème d'index

```

1 def entourage_case(terrain: list, x: int, y: int) -> (list):
2     """Retourne l'entourage d'une case donnée dans un terrain donné.
3
4 Arguments:
5 - terrain      - list - le terrain dans lequel agir,
6 - x            - int - la coordonnée x de la case dont on veut l'entourage,
7 - y            - int - la coordonnée y de la case dont on veut l'entourage.
8
9 Retourne:
10 - entourage    - list - la liste contenant l'entourage de la case."""
11
12     # Si y-1 ou x-1 = -1, on risque d'accéder au mauvais élément de terrain
13     # lors de la vérification de l'entourage de la case donc on le place
14     # volontairement en dehors des possibilités d'accès afin de lever une
15     # erreur et d'ajouter une CASE au lieu de recompter une MINE lors de
16     # la vérification
17     y_moins = y - 1 if y - 1 >= 0 else len(terrain)
18     x_moins = x - 1 if x - 1 >= 0 else len(terrain[0])
19
20     # L'entourage de la case vérifiée
21     entourage = []
22
23     # Haut, à gauche
24     try: entourage.append(terrain[y_moins][x_moins])
25     except IndexError: entourage.append(CASE)
26
27     # Haut, au milieu
28     try: entourage.append(terrain[y_moins][x])
29     except IndexError: entourage.append(CASE)
30
31     # Haut, à droite
32     try: entourage.append(terrain[y_moins][x+1])
33     except IndexError: entourage.append(CASE)
34
35     # Milieu, à droite
36     try: entourage.append(terrain[y][x+1])
37     except IndexError: entourage.append(CASE)
38
39     # Bas, à droite
40     try: entourage.append(terrain[y+1][x+1])
41     except IndexError: entourage.append(CASE)
42
43     # Bas, au milieu
44     try: entourage.append(terrain[y+1][x])
45     except IndexError: entourage.append(CASE)
46
47     # Bas, à gauche
48     try: entourage.append(terrain[y+1][x_moins])
49     except IndexError: entourage.append(CASE)
50
51     # Milieu, à gauche
52     try: entourage.append(terrain[y][x_moins])
53     except IndexError: entourage.append(CASE)
54
55     return entourage

```

Listing 5 – Fonction *entourage_case* finale

```

1 def canvas_plateau(racine: tk.Tk,
2                     largeur: int, hauteur: int,
3                     col=0, lig=0) -> (None):
4     """Dessine le plateau de base du jeu, avant que l'utilisateur ne
5     commence \
6     à jouer.
7     Argument:
8     - racine          - tk.Tk - la fenêtre dans laquelle dessiner le plateau,
9     - largeur         - int - largeur en case du terrain,
10    - hauteur         - int - hauteur en case du terrain,
11    - col=0           - int - la colonne de la racine où sera placé le plateau,
12    - lig=0           - int - la ligne de la racine où sera placé le plateau.
13
14    Modifie:
15    - cv_plateau      - tk.Canvas - le plateau du jeu."""
16
17    global cv_plateau, cases_taille, cases, cases_pos
18
19    # On nettoie le plateau précédent, on remet à zéro les variables en
20    # ayant
21    # besoin
22    cv_plateau.destroy()
23    cases = {}
24    cases_pos = {}
25
26    # On met à jour les images
27    maj_taille_cases(largeur, hauteur)
28    maj_images(cv_plateau, cases_taille)
29
30    # Initialise le nouveau plateau
31    cv_plateau = tk.Canvas(racine,
32                           width=largeur * cases_taille,
33                           height=hauteur * cases_taille)
34
35    # Place le plateau
36    cv_plateau.grid(column=col, row=lig, padx=10, pady=10)
37
38    # Dessine les cases du jeu
39    for y in range(hauteur):
40        for x in range(largeur):
41            # Récupère la référence en dessinant la case
42            case = label_case(cv_plateau, x, y)
43            # Associe la référence de la case à ses coordonnées (x, y)
44            cases[case] = (x, y)
45            # Associe les coordonnées (x, y) à la référence de la case
46            cases_pos[(x, y)] = case

```

Listing 6 – Fonction *canvas_plateau*

Problèmes du *Garbage Collector* :

```
1 cases_img = {
2     0: tk.PhotoImage(file='%scase_0.gif' % chemin),
3     1: tk.PhotoImage(file='%scase_1.gif' % chemin),
4     2: tk.PhotoImage(file='%scase_2.gif' % chemin),
5     3: tk.PhotoImage(file='%scase_3.gif' % chemin),
6     4: tk.PhotoImage(file='%scase_4.gif' % chemin),
7     5: tk.PhotoImage(file='%scase_5.gif' % chemin),
8     6: tk.PhotoImage(file='%scase_6.gif' % chemin),
9     7: tk.PhotoImage(file='%scase_7.gif' % chemin),
10    8: tk.PhotoImage(file='%scase_8.gif' % chemin),
11    BASE: tk.PhotoImage(file='%sbase.gif' % chemin),
12    DRAPEAU: tk.PhotoImage(file='%sdrapeau.gif' % chemin),
13    MINE: tk.PhotoImage(file='%smine.gif' % chemin),
14    PERDU: tk.PhotoImage(file='%sperdu.gif' % chemin)
15 }
```

Listing 7 – Version 1 de *cases_img*

```
1 cases_img = {
2     0: tk.PhotoImage(master=racine, file='%scase_0.gif' % chemin),
3     1: tk.PhotoImage(master=racine, file='%scase_1.gif' % chemin),
4     2: tk.PhotoImage(master=racine, file='%scase_2.gif' % chemin),
5     3: tk.PhotoImage(master=racine, file='%scase_3.gif' % chemin),
6     4: tk.PhotoImage(master=racine, file='%scase_4.gif' % chemin),
7     5: tk.PhotoImage(master=racine, file='%scase_5.gif' % chemin),
8     6: tk.PhotoImage(master=racine, file='%scase_6.gif' % chemin),
9     7: tk.PhotoImage(master=racine, file='%scase_7.gif' % chemin),
10    8: tk.PhotoImage(master=racine, file='%scase_8.gif' % chemin),
11    BASE: tk.PhotoImage(master=racine, file='%sbase.gif' % chemin),
12    DRAPEAU: tk.PhotoImage(master=racine, file='%sdrapeau.gif' % chemin),
13    MINE: tk.PhotoImage(master=racine, file='%smine.gif' % chemin),
14    PERDU: tk.PhotoImage(master=racine, file='%sperdu.gif' % chemin)
15 }
```

Listing 8 – Version finale de *cases_img*

Note Les variables notées en majuscules, à l’instar de *BASE*, sont des constantes issues du fichier *constantes.py*, qui ne sert à rien d’autre qu’à les enregistrer.

```

1 def maj_revele_case(x: int, y: int) -> (None):
2
3     global cases, cases_pos, cases_img
4
5     # On récupère la valeur de la case et on continue dans la fonction
6     # uniquement si la case n'a pas déjà été révélé
7     try: valeur_case = joueur.terrain[y][x]
8     except IndexError: return
9     finally:
10         if (x, y) in joueur.cases_vues:
11             return
12
13     # On récupère la case sous sa forme de tk.Label
14     case = cases_pos[(x, y)]
15
16     # Si la case est un chiffre basique, on l'affiche
17     if 0 < valeur_case < 9:
18         joueur.cases_vues.append((x, y))
19         case['image'] = cases_img[valeur_case]
20     # Si la case vaut zéro, on révèle l'entourage
21     elif valeur_case == 0:
22         joueur.cases_vues.append((x, y))
23         case['image'] = cases_img[valeur_case]
24
25         # Pour éviter les indices négatifs et les problèmes qui vont avec,
26         # on provoque une IndexError dans les appels suivants
27         y_moins = y - 1 if y - 1 >= 0 else len(joueur.terrain)
28         x_moins = x - 1 if x - 1 >= 0 else len(joueur.terrain[0])
29
30         # On teste les cases autour, pour les révéler ou propager la
31         # révélation
32         # si elles sont de valeur 0
33         # Haut, à gauche
34         maj_revele_case(x_moins, y_moins)
35         # Haut, au milieu
36         maj_revele_case(x, y_moins)
37         # Haut, à droite
38         maj_revele_case(x+1, y_moins)
39         # Milieu, à droite
40         maj_revele_case(x+1, y)
41         # Bas, à droite
42         maj_revele_case(x+1, y+1)
43         # Bas, au milieu
44         maj_revele_case(x, y+1)
45         # Bas, à gauche
46         maj_revele_case(x_moins, y+1)
47         # Milieu, à gauche
48         maj_revele_case(x_moins, y)
49     # Si la case n'était pas chiffrée (une mine donc), on la révèle
50     else:
51         joueur.cases_vues.append((x, y))
52         # Affiche la mine de couleur rouge si c'est celle qui fait perdre la
53         # partie, sinon la mine noire
54         if not ordi.fini: case['image'] = cases_img[PERDU]
55         else: case['image'] = cases_img[MINE]
56
57     ordi.nb_cases_vues += 1
58     ordi.verif_etat_partie(valeur_case)

```

Listing 9 – Fonction *maj_revele_case*