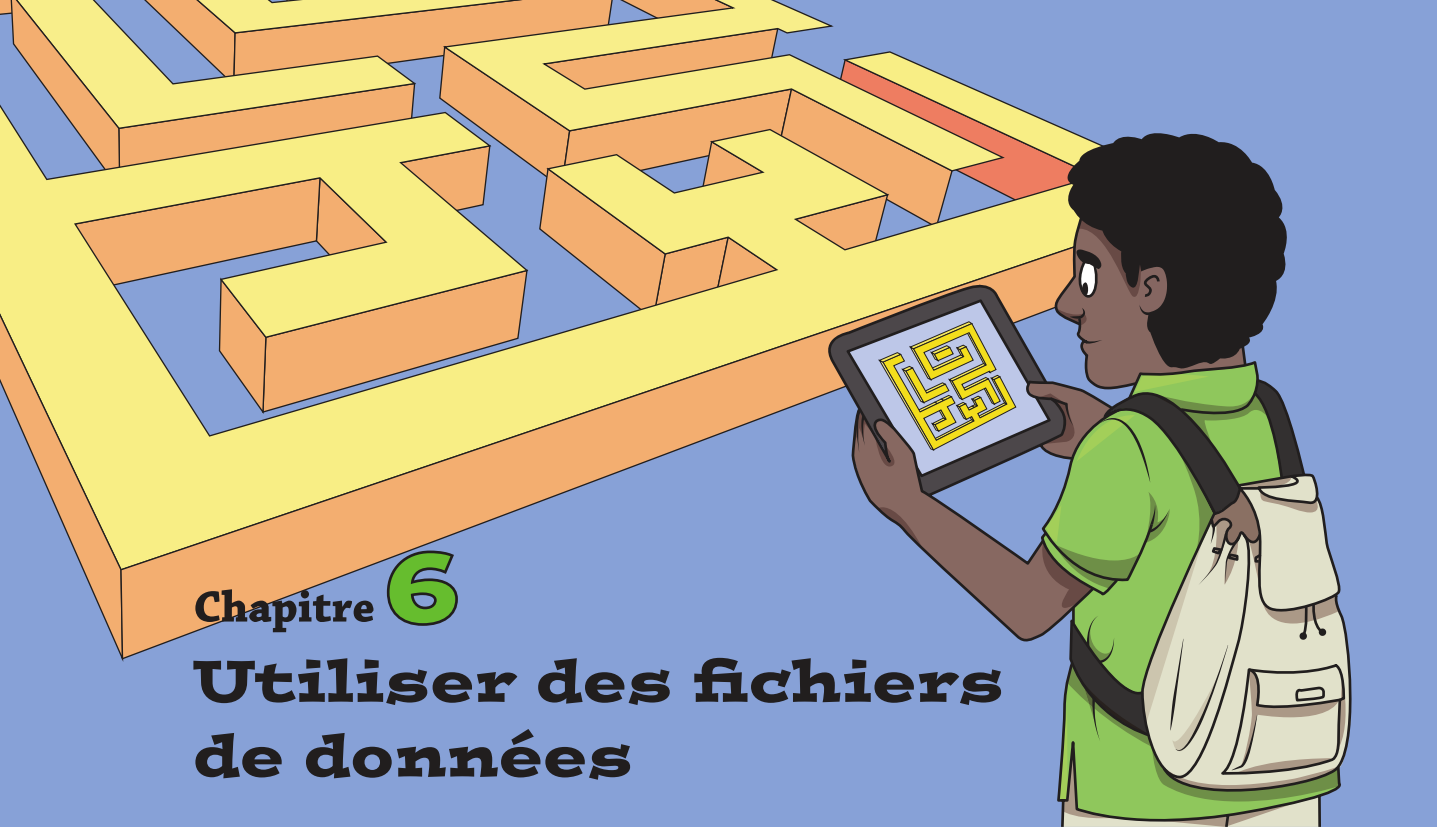

Chapitre 6

Utiliser des fichiers de données

C'EST LORSQUE TU commences à manipuler d'importantes quantités de données que la programmation trouve tout son intérêt. Ton programme devient alors une série de règles qui déterminent la manière dont ces données vont être lues et traitées et la forme qu'elles vont prendre. C'est à partir de là que les données vont commencer à jouer un rôle important.

Dans ce chapitre, tu vas d'abord apprendre à créer des fichiers de texte qui vont te servir à concevoir des labyrinthes. Cela te permettra de construire automatiquement des labyrinthes dans Minecraft pour que tes amis et toi puissiez vous y perdre. Tu verras qu'avec un petit programme Python, il est possible de construire des structures aussi grandes que des labyrinthes. Tu n'en croiras pas tes yeux !

Tu vas ensuite t'inspirer de ton programme de création de labyrinthe pour en développer un autre beaucoup plus complet, qui te permettra de bâtir une gigantesque structure de scannage et d'impression en 3D, également appelée salle de reproduction. Tout ce que tu confectionneras à l'intérieur pourra être enregistré dans un fichier et réutilisé plus tard. Tu pourras notamment téléporter tes objets à des endroits différents dans Minecraft et même les charger sur d'autres ordinateurs pour les faire apparaître chez tes amis ! Tu te constitueras ton propre répertoire d'objets et pourras les reproduire si rapidement dans Minecraft que tous tes amis voudront eux aussi posséder une machine à dupliquer !

Lire une donnée dans un fichier

Un programme informatique n'est pas aussi intelligent qu'il en a l'air : il se contente bêtement de suivre les instructions qu'on lui donne. Si on ne change aucune donnée du programme, il s'exécutera toujours de la même manière. Les programmes Minecraft que tu as réalisés jusqu'à présent sont assez interactifs, car ils changent de comportement en fonction des événements qui surviennent dans le jeu. Ces événements constituent des données pour le programme.

L'intérêt des fichiers de données

Il existe une autre manière de changer le comportement d'un programme informatique chaque fois que tu l'exécutes : tu peux enregistrer des données dans un fichier texte et demander à ton ordinateur de le lire au démarrage. Tu peux alors créer tout un dossier de fichiers de texte et même rédiger un menu pour choisir le fichier à ouvrir en fonction de ce que tu souhaites réaliser avec le programme. On dit de ce type de programme qu'il est « orienté donnée », car ce sont principalement les données stockées dans le fichier de texte externe qui vont l'influencer.

En y réfléchissant, la plupart des programmes que tu utilises sur ton ordinateur font déjà appel à des fichiers de données. Le programme de traitement de texte que tu utilises pour faire tes devoirs enregistre ton travail dans un fichier de données ; lorsque tu prends une photo avec un appareil numérique, elle est stockée dans un fichier de données et ton programme d'édition de photo accède à ton image depuis ce fichier. Même Minecraft utilise des fichiers de données en coulisses, pour enregistrer ou charger ton monde par exemple, mais aussi pour les kits de textures qui servent à personnaliser tes blocs. Utiliser des fichiers de données dans un programme te permet de travailler en toute flexibilité, car tu peux ainsi lui demander d'exécuter tout un tas de tâches différentes, sans pour autant avoir besoin de le modifier à chaque fois pour y parvenir. L'avantage est que tu peux aussi partager ces fichiers de données avec tes amis, si toutefois ils possèdent le même programme que toi.

Le programme que j'ai publié sur mon blog il y a quelque temps est un bel exemple d'utilisation de fichiers de données dans Minecraft. Je l'ai baptisé le « Constructeur BMP de Minecraft » (`http://blog.whaleygeek.co.uk/minecraft-pi-with-python/`), car il permet de lire une image depuis un fichier bitmap (BMP) et de la reproduire bloc par bloc dans Minecraft. Grâce à ce programme, j'ai pu reproduire le logo de Raspberry Pi à l'identique dans le jeu. Il est tellement énorme que lorsqu'on se trouve à la base et qu'on lève la tête pour le regarder, on a du mal à croire qu'il s'agit du logo ! Ce programme est capable de lire n'importe quelle image BMP et de la reproduire à l'identique dans le jeu, sans qu'on ait besoin d'intervenir.

Lorsqu'on écrit un programme orienté données, il faut tout d'abord savoir ouvrir et lire des fichiers texte depuis son ordinateur à l'aide de Python.

Créer un donneur d'indices

Pour apprendre à ouvrir et lire des fichiers texte, tu vas devoir écrire un programme relativement simple de donneur d'indices. Ce programme va te permettre de lire un fichier texte contenant des trucs et astuces sur le jeu Minecraft et d'afficher ces conseils un par un sur le chat, de manière aléatoire. Pour cela, tu dois d'abord créer un fichier et y rédiger des indices. Tu peux le faire dans l'éditeur Python, comme pour les programmes.

Lance Minecraft, IDLE ainsi que le serveur si tu es sur un PC ou un Mac. Tu t'es déjà certainement un peu familiarisé avec le processus, mais n'hésite pas à relire le chapitre 1 si tu as besoin de te rafraîchir la mémoire.

1. Dans IDLE, crée un nouveau fichier de texte en cliquant sur `File>New File` dans le menu. Enregistre ton programme en le nommant `astuces.txt`.
2. Puis énumère quatre ou cinq astuces en plaçant chacune sur une ligne de texte. Ci-après, j'ai rédigé quelques exemples d'astuces Minecraft pour t'aider, mais cela sera beaucoup plus amusant si tu inventes tes propres astuces. N'oublie pas d'appuyer sur `Entrée` à la fin de chaque ligne, pour créer une **fin de ligne** à l'intérieur du fichier lorsque tu changes d'astuce. Tu en sauras davantage sur les fins de ligne un peu plus tard dans ce chapitre.

```
Construis-toi un abri avant que les monstres n'apparaissent ↩
et ne viennent te dévorer.
Utilise des blocs d'eau pour alimenter des canaux souterrains.
Construis une structure en sable et détruis la base.
Utilise les deux modes de vue à la première et à la ↩
troisième personne.
Appuie deux fois sur la barre d'espace pour voler dans le ciel.
```

3. Enregistre ce fichier. Ne l'exécute pas puisqu'il ne s'agit pas d'un programme Python mais d'un fichier de texte que tu vas demander à Python d'utiliser.

Une **fin de ligne** est un caractère spécial invisible que l'ordinateur utilise pour marquer la fin d'une ligne de texte.

On l'appelle aussi parfois un « retour chariot », qui date de l'époque où on utilisait encore les machines à écrire. Lorsqu'on voulait aller à la ligne, on utilisait le retour chariot, qui faisait revenir le chariot à gauche de la page, une ligne plus bas.

N'utilise pas de virgule dans ton fichier `astuces.txt`, ni dans aucun des messages que tu veux faire apparaître avec `mc.postToChat()`. L'API de Minecraft comporte un petit bug qui coupe les phrases à l'endroit où on a placé une virgule.

Maintenant que tu as créé ton fichier d'astuces, c'est-à-dire de données, il est temps de passer à l'écriture du programme qui va les traiter. Ce programme choisira une astuce au hasard dans le fichier et l'affichera au bout d'un certain temps dans le chat de Minecraft. Ainsi, pendant que tu joueras, tu verras s'afficher des conseils utiles.

1. Ouvre un nouveau fichier en cliquant sur `File>New File` et enregistre-le en le nommant `astuceChat.py`.

2. Importe les modules dont tu as besoin :

```
import mcpi.minecraft as minecraft
import time
import random
```

3. Puis connecte-toi au jeu Minecraft :

```
mc = minecraft.Minecraft.create()
```

4. Définis ensuite une constante avec le nom du fichier pour pouvoir l'échanger facilement par la suite avec un autre fichier :

```
NOMDEFICHIER = "astuces.txt"
```

5. Ouvre le fichier en mode de lecture (réfère-toi à l'encadré « Percer les secrets du code » pour obtenir plus de précisions sur cette étape) :

```
f = open(NOMDEFICHIER, "r")
```

6. Et lis toutes les lignes du fichier dans une liste que tu nommeras `astuces`. Souviens-toi que tu as déjà utilisé des listes dans les chapitres 4 (constructeur de pont magique) et 5 (électronique) :

```
astuces = f.readlines()
```

7. Puis ferme le fichier lorsque tu auras terminé de le lire. Dès lors que tu as fini d'utiliser un fichier, il faut toujours penser à le refermer, car s'il reste ouvert il ne pourra pas être utilisé par d'autres applications, telles que l'éditeur dans lequel tu as rédigé ton premier texte :

```
f.close()
```

8. Maintenant rédige une boucle de jeu avec des temps d'attente aléatoires, oscillant entre trois et sept secondes :

```
while True:
    time.sleep(random.randint(3,7))
```

9. Ensuite, dis au programme de choisir un message aléatoire dans la liste `astuces` et de l'afficher sur le chat. Pour cela, tu dois utiliser la fonction `strip()` (qui veut dire déshabiller ou dépouiller), pour te débarrasser de tous les caractères de fin de ligne

superflus. Tu en sauras plus sur les caractères de fin de ligne un peu plus tard, donc ne t'inquiète pas si tu ne comprends pas bien de quoi il retourne pour l'instant.

```
msg = random.choice(astuces)
mc.postToChat(msg.strip())
```

Enregistre ton programme et exécute-le. Que se passe-t-il ? À mesure que tu avances dans le monde de Minecraft et que tu joues, tu devrais voir une astuce s'afficher sur le chat au petit bonheur, comme sur la figure 6-1. Observe la manière dont les messages apparaissent. Tu verras qu'ils disparaîtront petit à petit.

FIGURE 6-1 Des astuces apparaissent de façon aléatoire sur le chat au fil du jeu.

DÉFI

Ajoute des astuces à ton fichier `astuces.txt` en t'aidant de ton expérience du jeu Minecraft. Donne ton fichier `astuces.txt` et ton programme `astuceChat.py` à un ami qui débute dans Minecraft afin qu'il puisse l'utiliser et réaliser des choses extraordinaires dans le jeu. Tu peux même écrire toute une série de fichiers `astuces.txt` sur des thèmes différents et contenant plus ou moins de détails selon le niveau des joueurs. Publie-les ensuite sur un petit site Internet et laisse-les en libre accès pour que d'autres joueurs puissent les utiliser, sans oublier de partager aussi le programme `astuceChat.py`.

PERCER LES SECRETS DU CODE

Dans le programme `astuceChat.py`, la ligne suivante renferme un petit mystère :

```
f = open(NOMDEFICHIER, "r")
```

La fonction `open()` permet d'ouvrir le fichier affecté à la constante `NOMDEFICHIER`. Mais que signifie le « `r` » à la fin ? Lorsque tu ouvres un fichier, tu dois préciser à la fonction `open()` le niveau d'accès que tu souhaites lui attribuer. Ici, le « `r` » (pour « read », qui signifie « lire » en anglais) te permet d'ouvrir le fichier en mode lecture uniquement. Si jamais tu essayes de modifier le texte du fichier, tu recevras un message d'erreur. Cette fonctionnalité te permet de protéger tes fichiers contre tout dysfonctionnement. Si tu veux ouvrir un fichier et le modifier, tu peux utiliser la lettre « `w` » (pour « write », qui signifie « écrire » en anglais) ; et si tu veux faire les deux à la fois, utilise « `rw` » (pour lire et écrire dans le fichier).

Enfin, que signifie le `f` ? Le `f` est en fait une autre variable, qui stocke ce qu'on appelle un descripteur de fichier. Un descripteur de fichier te permet de « saisir » un fichier. C'est une sorte de référence virtuelle du fichier qui se trouve dans le système de ton ordinateur. Si tu veux réaliser quelque chose sur un fichier ouvert, utilise la lettre `f` pour pouvoir t'y référer. Cette variable est très pratique, car comme toutes les autres variables que tu as utilisées auparavant, tu peux ouvrir plusieurs fichiers à la fois avec plusieurs variables de fichier, comme cela :

```
f1 = open("config.txt", "r")
f2 = open("niveaux.txt", "r")
f3 = open("score.txt", "rw")
```

Tu peux utiliser les variables `f1`, `f2` et `f3` dans la suite de ton script pour identifier correctement les fichiers que tu souhaites lire ou modifier.

Construire des labyrinthes à l'aide d'un fichier de données

Maintenant que tu sais comment demander à tes programmes de lire des fichiers de données, tu es prêt pour l'aventure. Tu sais déjà à quel point il est simple de placer des blocs dans le monde de Minecraft, grâce aux chapitres précédents. Mais que dirais-tu si tu pouvais construire des blocs en utilisant des listes de blocs répertoriées dans un fichier de données ? C'est exactement ce que tu vas faire ici : tu vas construire un grand labyrinthe en 3D dans Minecraft, en enregistrant les données de ton labyrinthe dans un fichier externe ! Mais avant tout, tu dois définir la manière dont tu vas organiser tes données dans le fichier.

Tu peux regarder une vidéo didactique sur la création d'un labyrinthe en te rendant sur le site Internet du livre www.editions-eyrolles.com/dl/14292. Choisis la vidéo « Adventure 6 » (en anglais).

Les labyrinthes que tu vas concevoir ici sont en 3D, car tu vas les construire dans un monde en 3D. Mais en réalité ce sont des labyrinthes en 2D construits à partir de blocs en 3D, car ils ne comportent qu'une seule couche. En clair, tu ne devras enregistrer que les données x et z de chaque bloc du labyrinthe dans ton fichier, pour que celui-ci soit rectangulaire.

Tu devras également décider de l'agencement de tes blocs dans le fichier de données. Enfin, comme tu n'utiliseras que des murs et de l'air, le chiffre 1 représentera les murs et le 0 représentera l'air. Tu pourras toujours modifier plus tard les types de blocs servant à la construction d'un mur à l'intérieur de ton programme Python.

Comprendre les fichiers CSV

Pour ce programme, tu vas utiliser un fichier texte un peu spécial, qui s'appelle **CSV**. Il te permettra de définir tes labyrinthes sous la forme de fichiers au sein du système de fichiers de ton ordinateur.

Un fichier **CSV** est un fichier qui contient des valeurs séparées par des virgules. On y écrit des valeurs sous forme de texte et on les sépare par des virgules. Dans un fichier CSV, il est possible de créer un tableau ou une base de données simple. Chaque ligne du fichier représente une rangée du tableau ou une donnée de la base de données ; et le texte des lignes entre les virgules représente une colonne du tableau ou un champ. Certains fichiers CSV utilisent la première ligne du fichier pour enregistrer les titres des colonnes du tableau ou les noms des champs. Toutes les applications de tableur et de base de données les plus connues sont compatibles avec les fichiers CSV.

Le format CSV est très simple et son utilisation est très répandue. Voici un exemple de texte CSV issu d'un tableau de base de données rassemblant des informations sur des joueurs de Minecraft. Dans ce fichier CSV, la première rangée (ou ligne) est réservée aux noms des champs et toutes les autres lignes comportent des données séparées par des virgules :

```
Nom,Pseudo,Spécialité
David,w_geek,Le code en Python
Roma,physics_gurl,La conception de grands bâtiments
Ryan,mr_teck,Les robots Minecraft
Craig,rrrrrrrr,Expert TNT
```

Ce fichier CSV contient donc une rangée avec les titres `Nom`, `Pseudo` et `Spécialité`. Il contient également quatre rangées de données, comportant chacune trois champs.

Lorsqu'on conçoit des structures complexes en informatique, il est parfois utile d'avoir recours à d'autres outils. En tant qu'ingénieur informatique, j'utilise régulièrement les tableurs pour concevoir et représenter des données. Ils constituent un formidable moyen de créer des tableaux de données, car on peut les exporter au format CSV et les charger ensuite dans d'autres applications. Pour t'entraîner, essaye de concevoir un labyrinthe dans ton application de tableur favorite. Crée des colonnes étroites et utilise les formules pour remplir tes cellules en blanc lorsque tu saisis le chiffre `0` et en jaune lorsque tu saisis le chiffre `1`. Une fois que tu as visualisé ton labyrinthe de cette manière, exporte-le au format CSV et nomme-le `labyrinthe.csv`, puis exécute ton programme et amuse-toi à te perdre avec tes amis !

Dans le fichier de données de ton labyrinthe, tu n'as pas besoin d'une rangée de titres, car chaque colonne du tableau comporte le même type de données, à savoir le chiffre 0 pour représenter le vide et le chiffre 1 pour représenter un morceau de mur. Tu peux télécharger le fichier du labyrinthe depuis le site Internet du livre, mais voici son contenu si tu préfères le créer toi-même.

1. Crée un nouveau fichier en cliquant sur `File>New File`. Clique sur `File>Save As` et nomme ton fichier `labyrinthe1.csv`.
2. Saisis les lignes suivantes en les recopiant méticuleusement et fais attention à conserver le même nombre de colonnes sur chaque ligne. Si tu observes ces données de près, tu devrais déjà pouvoir te faire une idée de la structure du labyrinthe ! Au total, il contient 16 lignes, avec 16 chiffres par ligne, alors tu peux t'aider d'un crayon pour vérifier que tu prends bien toutes les données en compte :

```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,1,1,1,1,1,1,1,1,0,1,0,1,1,0,1
1,0,0,1,0,0,0,0,1,0,1,0,1,0,0,1
1,1,0,1,0,1,1,0,0,0,0,0,1,0,1,1
1,1,0,1,0,1,1,1,1,1,1,1,1,0,1,1
1,1,0,0,0,1,1,1,1,1,0,0,0,0,1,1
1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1
1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1
1,0,1,1,1,1,0,0,0,0,0,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,1,1,1,1,1,1,1,1,0,1,1,1,1,1
1,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1
1,0,1,0,1,1,1,1,1,0,1,1,1,1,1,0,1
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1
1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1
```

3. Enregistre ce fichier. Tu l'utiliseras lorsque tu auras écrit le programme Python te permettant de lire et de traiter ces données.

Construire un labyrinthe

Maintenant que tu disposes d'un fichier de données détaillant la structure de ton labyrinthe, il ne te reste plus qu'à rédiger le programme Python qui va lire ces données et construire ton labyrinthe dans Minecraft avec des blocs.

1. Ouvre un nouveau fichier en cliquant sur `File>New File`, puis enregistre-le en cliquant sur `File>Save As` et nomme-le `constructionCSV.py`.
2. Importe les modules dont tu as besoin :

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

3. Et connecte-toi au jeu Minecraft :

```
mc = minecraft.Minecraft.create()
```

4. Définis quelques constantes. La constante `ESPACE` est le type de bloc que tu vas utiliser entre les murs. Normalement, tu y mettras de l'air, mais tu peux aussi essayer d'y inclure d'autres types de blocs pour rendre ton labyrinthe encore plus intéressant. La constante `MUR` est le type de blocs que tu vas utiliser pour créer les murs de ton labyrinthe. Tu dois donc faire attention à ce que tu choisis, car certains blocs ne permettent pas de bâtir des murs solides. Quant à la constante `SOL`, elle sert de base à ton labyrinthe :

```
ESPACE = block.AIR.id
MUR = block.GOLD_BLOCK.id
SOL = block.GRASS.id
```

N'utilise pas le `SABLE` pour le sol de ton labyrinthe. Si jamais tu poses du sable comme fondation, il se peut que le sol se dérobe sous les pieds de ton joueur en fonction du terrain sur lequel il se trouve !

5. Ouvre le fichier contenant les données de ton labyrinthe. Ici, tu vas utiliser la constante `NOMDEFICHIER`, pour te faciliter la tâche lorsque tu modifieras plus tard le nom du fichier afin de lire des données de labyrinthe différentes :

```
NOMDEFICHIER = "labyrinthe1.csv"
f = open(NOMDEFICHIER, "r")
```

6. Localise ton joueur et calcule les coordonnées de l'angle inférieur de ton labyrinthe. Ajoute 1 aux coordonnées x et z de ton joueur afin d'éviter de construire le

labyrinthe sur sa tête. Tu peux également t'amuser à construire le labyrinthe dans le ciel en modifiant la coordonnée y :

```
pos = mc.player.getTilePos()
ORIGINE_X = pos.x+1
ORIGINE_Y = pos.y
ORIGINE_Z = pos.z+1
```

7. Comme la coordonnée z variera à la fin de chaque ligne de données, commence par celle du point de départ de ton labyrinthe. Elle changera légèrement plus tard dans le programme.

```
z = ORIGINE_Z
```

8. Tu vas maintenant faire tourner une boucle sur toutes les lignes du fichier. Pour cela, lis attentivement l'encadré « Percer les secrets du code », qui t'explique toutes les étapes de la fonction `readlines()`. La boucle `for` te permet de lire chaque ligne du fichier une à une. À chaque tour de boucle, la variable `line` contient la donnée de la ligne qui vient d'être lue :

```
for line in f.readlines():
```

9. Tu vas maintenant diviser la ligne en plusieurs parties, là où sont placées les virgules. Pour en savoir plus sur la fonction `split()`, consulte l'encadré intitulé « Percer les secrets du code ». Souviens-toi que les lignes qui font partie du corps de la boucle `for` doivent être indentées d'un cadratin :

```
    data = line.split(",")
```

10. Fais bien attention à cette étape, car tu vas introduire une autre boucle `for`. C'est ce qu'on appelle une **boucle imbriquée**, car une boucle `for` est imbriquée dans une autre `for`. Dans ton programme, tu vas devoir réinitialiser la coordonnée x au début de chaque nouvelle rangée du labyrinthe. C'est pourquoi, tu dois le faire en dehors de la boucle `for` qui dessine une rangée entière :

```
    x = ORIGINE_X
    for cell in data:
```

11. Tous les chiffres que le programme va lire dans la ligne seront considérés comme du texte (une chaîne de caractères), donc tu devras placer des guillemets droits " autour de chacun pour que ton programme puisse fonctionner. L'instruction `if/else` te permet de construire soit un espace soit un mur en fonction du chiffre que le programme aura lu dans le fichier CSV. Pour rappel, le 0 correspond à un espace et toute autre donnée conduit à la construction d'un mur. Fais bien attention à l'indentation dans ce passage : l'instruction `if` est indentée deux fois, car elle fait partie de la boucle `for cell in data`, qui fait elle-même partie de la boucle `for line in f.readlines()`. La variable `b` est quant à elle très utile, car elle permet de raccourcir et de simplifier le programme. (Essaye de réécrire ce passage en omettant volontairement la variable `b` et tu verras ce qu'il se passe !)

```
        if cell == "0":
            b = ESPACE
        else:
            b = MUR
        mc.setBlock(x, ORIGINE_Y, z, b)
        mc.setBlock(x,ORIGINE_Y+1, z, b)
        mc.setBlock(x, ORIGINE_Y-1, z, SOL)
```

12. Mets la coordonnée x à jour à la fin de la boucle `for cell`. Cette ligne doit être indentée, car elle s'aligne sur l'instruction précédente `mc.setBlock()` et elle fait partie du corps de la boucle `for cell in data`.

```
        x = x + 1
```

13. Mets la coordonnée z à jour à la fin de la boucle qui traite chaque rangée de données. Ne l'indente que d'un niveau, car elle fait partie de la boucle `for line in f.readlines()`.

```
    z = z + 1
```

Enregistre ton programme et vérifie bien tous tes niveaux d'indentation. Si jamais tu te trompes, le programme ne fonctionnera pas comme prévu et tu obtiendras un labyrinthe vraiment très étrange !

Exécute ton programme et tiens-toi prêt à voir apparaître un superbe labyrinthe sous tes yeux. Parcoure ses allées et essaye d'en trouver la sortie sans briser les murs et sans t'envoler dans les airs. Sur la figure 6-2, tu peux voir à quoi il ressemble de l'intérieur.

FIGURE 6-2 Vue de l'intérieur du labyrinthe dans Minecraft

Si tu es égaré et que tu ne trouves pas la solution, tu auras toujours la possibilité de tricher un peu et de t'envoler pour voir ton labyrinthe d'en haut, comme sur la figure 6-3.

FIGURE 6-3 Vue aérienne du labyrinthe dans Minecraft

Une **boucle imbriquée** est une boucle à l'intérieur d'une autre boucle. On appellera la première une « boucle externe », et la deuxième une « boucle interne ». En Python, on imbrique une boucle dans une autre en l'indentant d'un niveau supplémentaire. On peut imbriquer plusieurs boucles les unes dans les autres autant de fois qu'on le souhaite.

Si tu laisses des lignes vides à la fin de ton fichier de données, elle seront lues et interprétées par ton programme Python. Heureusement, cela n'aura aucune incidence sur ton programme, mais il se peut que cela provoque la création de blocs en trop à la fin de ton labyrinthe.

PERCER LES SECRETS DU CODE

Dans les chapitres précédents, tu as appris à utiliser les listes Python et tu sais désormais qu'elles peuvent regrouper une série d'objets à des emplacements numérotés. Plusieurs fonctions inhérentes au langage Python fournissent

des données sous forme de liste, comme tu viens de le voir dans le constructeur de labyrinthe.

Cette ligne de code en est l'exemple :

```
for line in f.readlines():
```

Souviens-toi que *f* est un descripteur de fichier, c'est-à-dire quelque chose dont tu peux t'emparer pour accéder à un fichier ouvert. Le type `file` dans Python possède une fonction intégrée qui s'appelle `readlines()`. Elle permet simplement de lire chaque ligne du fichier et de les accoler une à une à la fin de la liste. Tu dois également te souvenir, grâce aux chapitres précédents, que lorsqu'on inclut une liste dans une boucle `for`, celle-ci passe en revue tous ses objets.

Tout ce que la boucle `for` fait, c'est qu'elle lit toutes les lignes d'un fichier qui se trouvent au sein d'une liste et qu'elle les passe en revue l'une après l'autre. À chaque tour de boucle, la variable `line` (c'est-à-dire la variable de contrôle de boucle) s'empare de la ligne de données suivante.

Par exemple, si ton fichier contient trois lignes, comme ceci :

```
ligne un
ligne deux
ligne trois
```

la fonction `f.readlines()` renverra une liste ressemblant à ceci :

```
['ligne un', 'ligne deux', 'ligne trois']
```

Par ailleurs, tu as déjà utilisé une liste dans une autre partie de ton script, mais peut-être que tu ne t'en es pas aperçu. C'était ici :

```
data = line.split(",")
```

La variable `line` contient une chaîne de caractères qui correspond au texte d'une ligne complète du fichier CSV. Toutes les variables contenant une chaîne de caractères comportent une fonction intégrée qui s'appelle `split()`. Cette fonction divise la chaîne de caractères pour en faire une liste. Les caractères `","` entre parenthèses indiquent à la fonction `split` l'endroit où la séparation doit s'opérer.

Imagine que tu disposes d'une variable `line` comportant trois mots séparés par une virgule, comme ceci :

```
line = "un,deux,trois"
data = line.split(",")
```

Ta variable `data` comportera alors une liste de trois objets, ce qui donnera le résultat suivant :

```
['un','deux','trois']
```

DÉFI

Conçois ton propre labyrinthe dans un autre fichier CSV en dessinant beaucoup d'allées sinueuses, d'impasses et de spirales pour rendre la tâche encore plus ardue. Mets tes amis au défi de retrouver la sortie. Tu peux t'amuser à cacher des trésors çà et là (en utilisant par exemple un `DIAMOND_BLOCK`, c'est-à-dire un bloc de diamant) ainsi qu'à la sortie du labyrinthe pour que tes amis puissent prouver qu'ils en ont exploré toutes les allées. Tu peux même t'amuser à construire un labyrinthe gigantesque qui recouvre tout le monde de Minecraft ! Trouveras-tu le moyen d'enregistrer la position des trésors dans le fichier du labyrinthe ?

Lorsque j'ai conçu les programmes de ce chapitre, j'ai dû apporter de légères modifications au programme du labyrinthe, car il était parfois trop difficile à résoudre. Par exemple, que se passe-t-il si tu retires la ligne de code qui pose les fondations du labyrinthe ? Essaye de construire les murs en utilisant d'autres types de blocs, tels que le `CACTUS` ou l'`EAU` (`WATER`) et observe ce qu'il se passe. J'ai dû tester le programme à plusieurs reprises avant de trouver les types de blocs qui me convenaient pour la construction des murs et du sol !

DÉFI

Une erreur s'est glissée dans le programme et c'est à toi de la retrouver. Même si une ligne se termine par un 0 dans le programme du labyrinthe, il construira un mur en or à la place. D'après toi, quelle en est la raison ? Essaye de corriger ce défaut par toi-même. Indice : tu as déjà résolu un problème similaire dans le programme `astuceChat.py`.

Dans ce chapitre, tu as utilisé les fonctions `readlines()` et `split()` pour traiter des fichiers CSV. Python est un langage de programmation très vaste et déjà très reconnu, qui possède de nombreuses fonctions intégrées. Tu peux t'intéresser à son module intégré d'importation de CSV en cliquant sur `import csv`, car c'est une méthode encore plus puissante que celle que nous avons utilisée ici. En ce qui me concerne, je préfère développer mes programmes pas à pas, parce que cela m'aide à comprendre comment ils fonctionnent. C'est pourquoi j'ai utilisé les fonctions `readlines()` et `split()` dans ce livre.

Construire une imprimante de blocs en 3D

On s'est bien amusé à créer des labyrinthes, mais maintenant que tu sais utiliser des fichiers de données, bien d'autres possibilités vont s'ouvrir à toi ! Nous avons construit une structure avec deux types de blocs sur une seule couche, mais il serait dommage de se limiter à cela ! Pour créer ton imprimante et scanneur 3D, tu vas t'appuyer sur ton programme de construction de labyrinthe, le but étant de pouvoir créer des répliques d'arbres, de maisons et de bien d'autres objets encore, en seulement quelques clics ! En réalité, il s'agit d'un constructeur de blocs, mais j'aime bien l'idée de l'imprimante 3D. Car lorsque le programme se met en route, tu peux voir les structures se construire rangée par rangée, comme le papier qui sort d'une imprimante ou les objets qu'une imprimante 3D construit couche après couche.

Tu commences à te débrouiller plutôt bien en programmation et chaque nouveau script que tu écris est toujours plus compliqué que les précédents. Comme dans le chapitre précédent, où tu as rédigé ton programme à partir de fonctions, tu vas développer ce programme pas à pas.

La technique qui consiste à transformer les différentes fonctionnalités d'un programme en fonctions et à les rassembler ensuite dans un script plus grand est très courante chez les développeurs professionnels. Elle part du principe qu'un grand programme n'est qu'une suite de petits programmes. Si la structure de ton grand programme est correcte dès le départ, tu pourras approfondir tes fonctions pas à pas tout en développant et testant ton programme. En général, j'aime bien me créer des petites démos simples et rapides pour pouvoir montrer aux curieux ce que mon programme est capable de réaliser.

Fabriquer un petit objet de test pour imprimer en 3D

Dans ton programme de labyrinthe, tu as enregistré des données dans un fichier CSV dans lequel chaque ligne, ou rangée, correspond à une ligne de blocs dans Minecraft. Chaque rangée comporte elle-même des colonnes (séparées par des virgules) qui correspondent à un bloc de la ligne. Il s'agit là d'une structure en 2D, car elle pose un bloc uniquement sur les coordonnées x et z du rectangle où tu as choisi de construire ton labyrinthe. Même s'il apparaît dans le monde de Minecraft et qu'il est fait de cubes, le labyrinthe que tu as créé est en réalité en 2D, car il ne comporte qu'une seule couche de blocs.

Pour construire des formes en 3D, tu vas devoir ajouter la dimension y à ton programme. Si tu conçois qu'un objet en 3D est une succession de couches, le programme que tu vas développer n'est pas tellement plus compliqué que celui que tu as écrit pour bâtir ton labyrinthe. Pour créer un cube dont les faces mesurent cinq blocs de longueur et cinq de largeur, il te suffit d'enregistrer cinq couches d'informations.

Seulement voilà, ce n'est pas aussi simple que cela. Ton programme Python ne connaît pas à l'avance le nombre de lignes que le fichier contient. Il pourrait éventuellement en faire une estimation approximative, mais il serait préférable de concevoir un format de fichier CSV assez flexible pour que le programme puisse appréhender des objets de tailles différentes.

Pour résoudre ce problème, tu vas ajouter ce que l'on appelle des **métadonnées** à la première ligne du fichier, qui détaillent les dimensions de l'objet que tu vas construire.

Les **métadonnées** sont des données sur des données. Si ton fichier de données contient des types de blocs servant à construire des objets, tu peux par exemple apporter des précisions sur les dimensions des objets, leur nom ou le nom de leur créateur en guise de métadonnées, c'est-à-dire les informations qui concernent ces données. De même que les métadonnées d'une photo enregistrée sur ton ordinateur définissent la date et l'heure à laquelle elle a été prise ou même le nom de l'appareil photo, les métadonnées de ton imprimante 3D te permettront d'obtenir tout un tas d'informations utiles sur tes objets en 3D.

Tu vas maintenant bâtir une petite grotte en 3D, que tu peux voir à la figure 6-4, pour t'exercer à tester les capacités d'impression en 3D de ton programme. Tu peux télécharger ce petit objet depuis le site Internet des éditions Eyrolles (`www.editions-eyrolles.com/dl/14292`) ou saisir les instructions qui vont suivre, comme tu l'as fait plus tôt avec le labyrinthe.

FIGURE 6-4 Tu as bâti une petite grotte à l'aide d'un fichier CSV.

Fais bien attention à laisser des lignes vides entre les différentes couches de ton objet. Elles te permettent de voir où commence chacune d'elles. De plus, le programme Python que tu écris s'attend à tomber sur des lignes vides, alors si tu les omets il ne fonctionnera pas correctement.

N'hésite pas à utiliser les fonctions copier-coller dans les étapes suivantes pour éviter de saisir trop de code à la main et ainsi limiter les risques d'erreur.

1. Ouvre un nouveau fichier en cliquant sur `File>New File` et enregistre-le en le nommant `objet1.csv`.

2. Saisis la première ligne de métadonnées relative aux dimensions de l'objet. Ton objet de test aura les dimensions d'un cube de 5 × 5 blocs, le x correspondant à la largeur, le y à la hauteur et le z à la profondeur :

```
5,5,5
```

3. Saisis ensuite la première couche d'épaisseur de ton objet test, en t'assurant de laisser une ligne vide avant la première ligne de données. Ta première couche de blocs correspond aux fondations de l'objet ; tu dois donc remplir toute la surface en saisissant à chaque fois le chiffre 1 qui correspond à l'emplacement de chacun des blocs :

```
1,1,1,1,1
1,1,1,1,1
1,1,1,1,1
1,1,1,1,1
1,1,1,1,1
```

4. Le cube que tu construis est vide à l'intérieur et dispose d'une ouverture, tu dois donc dessiner trois couches avec ce format-là. N'oublie pas de laisser une ligne vide avant chacune des trois séries de chiffres :

```
1,1,1,1,1
0,0,0,0,1
0,0,0,0,1
0,0,0,0,1
1,1,1,1,1
```

5. Copie cette série de chiffres et colle-la deux fois à la suite en t'assurant de laisser une ligne vide avant chacune.

6. Puis termine en recouvrant ton objet d'une couche solide. Pour cela, copie les rangées de données de l'étape 3, en laissant une ligne vide devant sa première rangée de chiffres.

Enregistre ton fichier sans l'exécuter. De toute façon, comme il ne s'agit pas d'un fichier Python, tu ne peux pas l'exécuter. En revanche, pour l'utiliser tu devras créer un programme Python et c'est ce que nous allons faire dans la prochaine étape de ce chapitre.

Rédiger le code de l'imprimante 3D

Maintenant que tu as fini d'écrire tes données de test, il est temps que tu te mettes à l'écriture du programme de l'imprimante 3D qui te permettra de construire la grotte dans le monde de Minecraft. Ce script ressemble beaucoup au programme du labyrinthe, à cela près qu'il contient trois boucles imbriquées, c'est-à-dire une boucle pour chacune des coordonnées x, y et z.

Ce programme est sans doute le plus élaboré que tu as écrit depuis le début de ce livre. Prends le temps de l'écrire étape par étape pour le réussir du premier coup et tu seras d'autant plus satisfait du résultat ! Si tu sens que tu bloques à un certain point, commence par vérifier avec attention toutes tes indentations. Et si tu es toujours bloqué après cela, n'oublie pas que tu as la possibilité de télécharger tous les programmes de ce livre sur le site du livre à l'adresse suivante : `www.editions-eyrolles.com/dl/14292`.

1. Ouvre un nouveau fichier en cliquant sur `File>New File`, puis enregistre-le en cliquant sur `File>Save As` dans le menu et nomme-le `impression3D.py`.

2. Importe les modules dont tu as besoin :

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

3. Connecte-toi au jeu Minecraft :

```
mc = minecraft.Minecraft.create()
```

4. Crée une constante en lui attribuant le nom de ton fichier de données. Cela te facilitera la tâche pour plus tard lorsque tu liras plusieurs fichiers contenant plusieurs objets :

```
NOMDEFICHIER = "objet1.csv"
```

5. Définis une fonction `impression3D()`. Cette fonction te servira plus tard pour la réalisation de ton projet final, donc veille à la nommer correctement :

```
def impression3D(nomdefichier, originex, originey, originez):
```

6. Comme tu l'as fait pour ton programme de construction de labyrinthe, ouvre le fichier de données en mode lecture et lis toutes les lignes dans une liste Python :

```
    f = open(nomdefichier, "r")
    lignes = f.readlines()
```

7. Le premier élément de ta liste se trouve à l'index 0. Il s'agit de la première ligne de ton fichier, celle qui regroupe les métadonnées. Divise cette ligne en plusieurs parties en utilisant la fonction `split()`. Enregistre ensuite ces trois nombres dans les variables `dimensionsx`, `dimensionsy` et `dimensionsz`. Dans cette étape, tu dois utiliser la fonction `int()`, car lorsque tu lis des lignes dans un fichier, elles sont interprétées comme des chaînes de caractères. Or, tu as besoin de les lire comme des nombres, car, plus tard, tu les utiliseras pour effectuer des calculs.

```
    coords = lignes[0].split(",")
    dimensionsx = int(coords[0])
    dimensionsy = int(coords[1])
    dimensionsz = int(coords[2])
```

8. Définis une variable `idxligne` pour enregistrer l'index de la ligne dans la liste `lignes[]` que tu utilises. Comme ton programme va parcourir la liste `lignes` en lisant les données correspondant aux différentes couches de ton objet 3D, tu ne peux pas utiliser cette variable comme variable de contrôle de boucle :

```
    idxligne = 1
```

9. La première boucle `for` parcourt chacune des couches verticales des données lues dans le fichier. Les premières lignes du fichier correspondent à la couche qui recouvre la base de l'objet, c'est-à-dire au niveau le plus bas des coordonnées y. Pour que les étapes du processus d'impression soient visibles dans Minecraft, tu peux ajouter une fonction `postToChat()` à cet endroit :

```
    for y in range(dimensionsy)
        mc.postToChat(str(y))
```

10. Ignore les lignes vides entre chaque couche dans ton fichier en ajoutant le chiffre 1 à la variable `idxligne`. Comme tu le sais, ces lignes vides servent uniquement à rendre la lecture du fichier plus agréable, ton programme doit les ignorer.

```
        idxligne = idxligne + 1
```

11. Commence une boucle imbriquée `for` qui lira la prochaine ligne et la divisera en plusieurs parties après chaque virgule. Fais bien attention à l'indentation ici ; la boucle `for` doit être indentée de deux cadratins (un pour la fonction et un pour

la boucle `for y`) et le code à l'intérieur de la boucle `for x` doit être indenté de trois cadratins.

```
        for in range(dimensionsx):
            ligne = lignes[idxligne]
            idxligne = idxligne + 1
            donnee = ligne.split(",")
```

12. Maintenant, commence la rédaction de ta troisième boucle `for` qui va parcourir chacun des blocs de la ligne qui vient d'être traitée et les construira dans Minecraft. La boucle `for z` est indentée de trois cadratins et le corps de la boucle doit être indenté quatre fois, donc sois bien vigilant en rédigeant ces lignes, sinon tes objets auront une forme bien étrange !

```
        for z in range(dimensionsz):
            blockid = int(donnee[z])
            mc.setBlock(originex+x, originey+y, originez+z, ↩
blockid)
```

13. Enfin, saisis les lignes du programme principal, qui ne doivent comporter aucune indentation. Elles vont lire la position de ton joueur et vont ensuite commander à la fonction `impression3D()` d'imprimer ton objet en 3D juste à côté de lui :

```
pos = mc.player.getTilePos()
impression3D(NOMDEFICHIER, pos.x+1, pos.y, pos.z+1)
```

Sauvegarde ton programme et exécute-le. Que vois-tu ? Promène-toi un peu dans le monde de Minecraft et exécute le programme. À chaque fois, tu devrais voir apparaître une grotte en pierre juste à côté de ton joueur. Sur la figure 6-5, tu peux voir comme il est simple de construire plusieurs grottes en pierre en un claquement de doigts.

DÉFI

Crée d'autres fichiers CSV contenant des objets 3D, modifie la constante `NOMDEFICHIER` dans le script `impression3D.py` et exécute-le pour disséminer des objets un peu partout dans le monde de Minecraft. Comme tes objets sont de plus en plus sophistiqués, tu peux t'amuser à imaginer plusieurs techniques de conception avant d'enregistrer les chiffres dans le fichier. Tu peux par exemple les dessiner dans un tableur ou encore te procurer une grande boîte de cubes en plastique pour les construire en dur puis les reprendre couche après couche pour indiquer les numéros de bloc dans un fichier CSV.

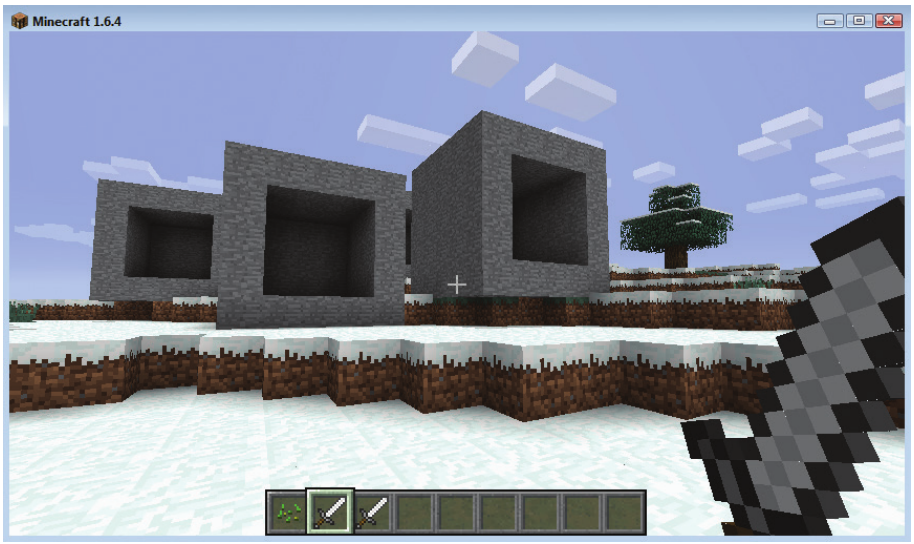

FIGURE 6-5 Impression 3D de plusieurs grottes en pierre dans Minecraft

Construire un scanneur de blocs en 3D

Ton imprimante 3D est déjà très puissante. Tu peux créer une grande bibliothèque de fichiers CSV pour construire tout un tas d'objets différents. Et dès que tu as besoin d'un objet en particulier, il te suffit d'exécuter ton programme `impression3D.py` avec la constante `NOMDEFICHIER` qui correspond à l'objet de ton choix pour le faire apparaître dans le monde de Minecraft.

En revanche, si tu souhaitais construire des objets plus complexes et plus grands, il te faudrait saisir tous les chiffres dans le fichier CSV à la main et cette tâche s'avérerait bien fastidieuse. Heureusement, le jeu Minecraft est idéal pour construire des objets complexes. Alors, pourquoi ne pas lui demander de créer lui-même ces fichiers CSV pour toi ? Dans ce chapitre, tu vas apprendre à créer une copie identique d'un arbre de ton choix pour la reproduire un peu partout dans le monde de Minecraft. Présentée ainsi, l'opération semble difficile à réaliser, mais il n'en est rien ! Car scanner en 3D, c'est un peu comme imprimer en 3D à l'envers.

1. Ouvre un nouveau fichier en cliquant sur `File>New File`, puis enregistre-le en cliquant sur `File>Save As` et nomme-le `scannage3D.py`.
2. Importe les modules dont tu as besoin :

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

3. Connecte-toi au jeu Minecraft :

```
mc = minecraft.Minecraft.create()
```

4. Crée des constantes avec les noms des fichiers CSV et la taille de la zone que tu souhaites scanner :

```
NOMDEFICHIER = "arbre.csv"
DIMENSIONSX = 5
DIMENSIONSY = 5
DIMENSIONSZ = 5
```

5. Définis une fonction `scannage3D()`. Tu l'utiliseras plus tard dans un programme, donc assure-toi de la nommer correctement :

```
def scannage3D(nomdefichier, originex, originey, originez)
```

6. Dans cette fonction, ouvre le fichier de données que tu vas créer, mais cette fois-ci en activant le mode de modification, c'est-à-dire en ajoutant `"w"` à l'intérieur de la fonction `open()` :

```
    f = open(nomdefichier, "w")
```

7. La première ligne du fichier doit contenir les métadonnées, tu dois donc y indiquer les valeurs des coordonnées x, y et z. Si tu veux connaître la signification et l'utilité du suffixe `\n` à la fin de la ligne de code, rends-toi dans l'encadré « Percer les secrets du code ».

```
    f.write(str(DIMENSIONSX) + "," + str(DIMENSIONSY) + "," ↩
+ str(DIMENSIONSZ) + "\n")
```

8. Le programme est l'exact opposé du programme d'impression et possède lui aussi trois boucles imbriquées. Je l'ai affiché dans son ensemble afin que l'indentation ressorte mieux et que tu puisses la retranscrire correctement. Pour en savoir plus sur la manière dont les lignes séparées par des virgules sont créées, lis l'encadré « Percer les secrets du code » :

```
    for y in range(DIMENSIONSY):
        f.write("\n")
        for x in range(DIMENSIONSX):
            line = ""
            for z in range(DIMENSIONSZ):
                blockid = mc.getBlock(originex+x, ↩
originey+y, originez+z)
                if line != "":
                    line = line + ","
                line = line + str(blockid)
            f.write(line + "\n")
        f.close()
```

9. Pour finir, le programme principal ne contient aucune indentation. Ces dernières lignes détectent ta position, calculent un espace cubique autour de ton joueur et demandent ensuite à la fonction `scannage3D()` de scanner tout cet espace pour l'inclure dans ton fichier CSV.

```
pos = mc.player.getTilePos()
scannage3D(NOMDEFICHIER, pos.x-(DIMENSIONSX/2), pos.y, ↩
pos.z-(DIMENSIONSZ/2))
```

Enregistre ton programme. Vérifie à nouveau que tes lignes de code sont bien indentées avant de l'exécuter.

Maintenant, cours vers un arbre et appuie-toi contre son tronc, puis lance ton programme `scannage3D.py`. Que se passe-t-il ?

Ne vois-tu rien se produire ? C'est tout à fait normal, souviens-toi que ce programme scanne un objet à l'intérieur d'un fichier CSV, donc tu dois regarder dans ton fichier `arbre.csv` pour voir quels chiffres il contient désormais. Ouvre le fichier `arbre.csv` en cliquant sur `File>Open` dans l'éditeur. La figure 6-6 montre une section du fichier `arbre.csv` que j'ai capturée après avoir embrassé un arbre ! Comme l'espace de scannage que j'ai défini est assez petit, il se peut que tu ne scannes qu'une petite partie de l'arbre que tu as choisi. Tu peux remédier à cela en modifiant les constantes `DIMENSIONS` de ton script pour scanner une zone plus grande.

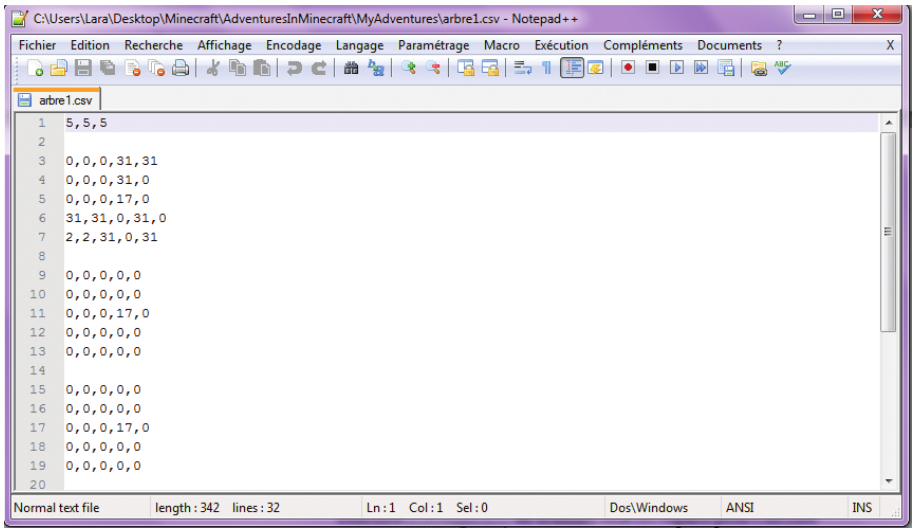

FIGURE 6-6 Contenu d'un fichier CSV après scannage d'un arbre

N'oublie pas que la direction dans laquelle ton joueur se tourne ne correspond pas forcément à la position qui a été scannée, donc regarde bien tout autour de toi. Le scanneur 3D numérise une zone comprise entre la position de ton joueur et des valeurs de coordonnées plus élevées. Donc, si ton joueur se trouve à l'emplacement 0,0,0, le scanneur 3D numérisera une zone comprise entre ces coordonnées et des coordonnées plus élevées, par exemple 4,4,4. Réfère-toi au schéma du chapitre 2, qui explique le principe des coordonnées et la direction que chacune d'entre elles représente.

Normalement, IDLE n'ouvre que les fichiers dotés de l'extension `.py` dans ses fenêtres `Open` et `Save`, mais tu peux aussi visualiser tes fichiers `.csv` en cliquant sur `Files of Type>All Files`.

DÉFI

Maintenant que tu disposes d'un scanneur et d'une imprimante 3D, modifie ton programme d'impression 3D pour utiliser une constante `NOMDEFICHIER` du fichier `arbre.csv`, puis promène-toi un peu partout dans Minecraft tout en exécutant le script `impression3D.py`. En faisant cela, tu devrais pouvoir parsemer des arbres sur ton chemin. Essaye d'imprimer des arbres dans le ciel et sous l'eau pour voir ce qui se passe. En combien de temps vas-tu parvenir à faire pousser toute une forêt ?

PERCER LES SECRETS DU CODE

Dans le programme `scannage3D.py`, tu as utilisé un caractère spécial au sein de la fonction `write()`. Cela mérite une petite explication :

```
f.write("\n")
```

Le caractère `\n` (appelé « backslash n ») est un caractère spécial non imprimable que Python te permet d'utiliser entre guillemets. Il signifie « passe à la prochaine ligne ». La lettre `n` est l'initiale de « newline » (en français, « nouvelle ligne »). Quant au backslash, la barre oblique inversée, il permet de distinguer cette lettre `n` d'une lettre n normale.

Dans le script scannage3D.py, tu as conçu ton fichier de sorte à laisser une ligne vide entre chacune des couches de données de ton objet pour faciliter la lecture et la modification du fichier CSV par d'autres personnes. L'instruction `f.write("\n")` réalise exactement la même chose : elle insère une ligne vide dans ton fichier.

Plus loin dans le programme scannage3D.py, tu peux t'amuser à rédiger du code autour de la variable `ligne`. En voici les instructions les plus importantes :

```
for x in range(DIMENSIONSX):
    ligne = ""
    for z in range(DIMENSIONSZ):
        blockid = mc.getBlock(originex+x, originey+y, ↩
  originez+z)
        if ligne != "": # la ligne n'est pas vide
            ligne = ligne + ","
        ligne = ligne + str(blockid)
```

Le code souligné correspond à un modèle de codage très répandu, généralement utilisé pour créer des valeurs séparées par des virgules. Au début de la boucle `x`, on affecte une chaîne de caractères vide à la variable. Chaque fois que la boucle `z` produit une autre valeur, elle vérifie d'abord si la première ligne est vide et, dans le cas contraire, elle ajoute une virgule avant d'ajouter les données d'identité du bloc. L'instruction `if` est nécessaire, car elle permet d'éviter qu'une virgule apparaisse en début de ligne, avant le premier nombre, ce qui pourrait perturber la lecture du programme impression3D.py par la suite.

construire une machine à dupliquer

Tu disposes maintenant de tous les scripts nécessaires pour développer ta propre machine à dupliquer en 3D, qui fera fureur chez tous tes amis ! Grâce à elle, tu pourras confectionner une salle à dupliquer dans le monde de Minecraft et créer tous les objets que tu veux. Tu pourras alors les conserver dans des fichiers, les réintroduire dans la salle pour les modifier ou les reproduire comme par magie un peu partout dans le monde. Enfin, tu pourras aussi t'évader de ce monde et faire disparaître la salle à dupliquer sans laisser aucune trace derrière toi.

À l'instar des chapitres précédents, tu vas utiliser les programmes que tu as créés et les assembler pour obtenir un programme plus grand. De même, comme ce programme possède plusieurs fonctionnalités, tu vas rédiger un menu pour qu'il soit plus facile à manipuler.

Rédiger la structure du programme de la machine à dupliquer

La première chose à faire est de concevoir la structure du programme pour qu'il tienne en un seul morceau. Pour cela, tu vas commencer par rédiger des fonctions factices qui ne feront qu'afficher leur nom lorsque tu les appelleras. Tu les compléteras par la suite avec le contenu des fonctions de tes autres programmes. C'est exactement la même technique que tu as employée pour développer ton jeu de chasse au trésor dans le chapitre 4, t'en souviens-tu ?

1. Ouvre un nouveau fichier en cliquant sur `File>New File` et enregistre-le en le nommant `duplicateur.py`.

2. Importe les modules dont tu as besoin :

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import glob
import time
import random
```

3. Connecte-toi au jeu Minecraft :

```
mc = minecraft.Minecraft.create()
```

4. Définis quelques constantes pour déterminer les dimensions de ta machine à dupliquer. Ne choisis pas des dimensions trop élevées, sinon ta machine à dupliquer tardera à scanner et à imprimer les objets. Pense également à définir la position par défaut de ta salle de reproduction :

```
DIMENSIONSX = 10
DIMENSIONSY = 10
DIMENSIONSZ = 10
sallex = 1
salley = 1
sallez = 1
```

5. Rédige quelques fonctions d'apparat correspondant aux fonctionnalités du programme. Tu les complèteras plus tard :

```
def constructionSalle(x, y, z):
    print("constructionSalle")

def demolitionSalle():
    print("démolitionSalle")

def nettoyageSalle():
    print("nettoyageSalle")

def repertoireFichiers():
```

```
    print("repertoireFichiers")

def scannage3D(nomdefichier, originex, originey, originez)
    print("scannage3D")

def impression3D(nomdefichier, originex, originey, originez):
    print("impression3D")
```

6. La fonction factice du menu est assez spéciale, car elle renvoie une valeur à la fin de son exécution. Pour l'instant, tu peux faire apparaître une option aléatoire et utiliser le mot-clé **return** pour tester les versions précédentes de ton programme. Tu reviendras dessus plus tard pour le compléter. Si tu rédiges un menu factice, c'est pour pouvoir tester la structure de ton programme. Tu le modifieras plus tard pour en faire un vrai menu :

```
def menu():
    print("menu")
    time.sleep(1)
    return random.randint(1,7)
```

7. Rédige la boucle de jeu principale qui va te permettre d'afficher le menu et d'utiliser la fonction correspondant à cette fonctionnalité. Pour en savoir plus sur l'utilisation de la variable `anotherGo`, lis l'encadré « Percer les secrets du code » :

```
anotherGo = True
while anotherGo:
    choix = menu()
    if choix == 1:
        pos = mc.player.getTilePos()
        constructionSalle(pos.x, pos.y, pos.z)
    elif choix == 2:
        repertoireFichiers()
    elif choix == 3:
        nomdefichier = raw_input("nom du fichier ?")
        scannage3D (nomdefichier, sallex+1, salley+1, sallez+1)
    elif choix == 4:
        nomdefichier = raw_input("nom du fichier ?")
        impression3D(nomdefichier, sallex+1, salley+1, sallez+1)
    elif choix == 5:
        scannage3D("scantemp", sallex+1, salley+1, sallez+1)
        pos = mc.player.getTilePos()
        impression3D("scantemp", pos.x+1, pos.y, pos.z+1)
    elif choix == 6:
        nettoyageSalle()
    elif choix == 7:
        demolitionSalle()
    elif choix == 8:
        anotherGo = False
```

Sauvegarde ton programme de test puis exécute-le. Que se passe-t-il ? Sur la figure 6-7, tu peux voir ce qui s'est affiché lorsque j'ai exécuté le programme sur mon ordinateur : il t'indique en effet les opérations qu'il effectue, c'est-à-dire qu'il choisit une entrée du menu de manière aléatoire, puis il exécute la fonction que cette entrée renferme. Comme les fonctions n'affichent que leur nom, c'est tout ce que tu peux voir pour l'instant. La suite de mots qui s'affichera sur ton écran sera différente de celle de la figure 6-7, car la fonction `menu()` choisit une entrée au hasard à chaque fois que ton programme y fait appel.

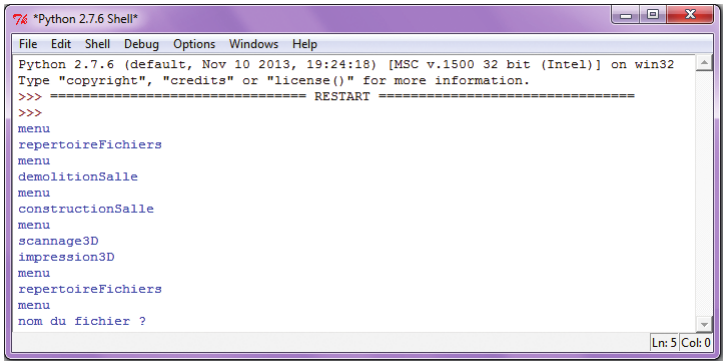

FIGURE 6-7 Ce qui s'affiche après l'exécution du programme de test

L'instruction **return** permet de renvoyer la valeur d'une fonction lorsqu'elle retourne dans le programme qui a appelé la fonction pour la première fois.

Par exemple, lorsque tu souhaites obtenir un nombre au hasard, tu appelles la fonction `random.randint()`. Cette fonction renvoie une valeur que tu peux alors stocker dans une variable, comme ceci : `a = random.randint(1,100)`. Le mot-clé `return` dans le langage Python te permet d'utiliser exactement la même technique de renvoi de valeurs au sein de tes propres fonctions.

La fonction `raw_input()` permet de lire une ligne de texte saisie à l'aide du clavier. Si tu saisis une chaîne de caractères entre les parenthèses, cette chaîne de caractères sera utilisée comme invite de commandes, donc l'instruction `nom = raw_input("Comment t'appelles -tu ?")` pose une question dont elle recevra également la réponse.

Lorsque tu utilises `raw_input()` pour lire du texte saisi au clavier, la fonction te renvoie toujours une chaîne de caractères. Si tu souhaites saisir un nombre (par exemple, dans ton menu), tu devras utiliser la fonction `int()` pour convertir la chaîne de caractères lue en nombre. Certains programmeurs Python préfèrent rédiger cette commande sur une seule et même ligne, comme ceci : `age = int(raw_input("Quel âge as-tu ?"))`.

Tout au long de ce livre, tu as utilisé la version 2 du langage Python. Si tu changes d'ordinateur, il se peut que la version 3 y soit installée. Ces deux versions du langage diffèrent sur plusieurs points. Par exemple, dans la version 3, tu dois utiliser la fonction `input()` au lieu de la fonction `raw_input()` propre à la version 2. Dans ce livre, j'ai choisi d'utiliser la version 2 du langage Python, car l'API de Minecraft est conçue pour fonctionner uniquement avec celle-ci. Si tu souhaitais utiliser la version 3, tu serais obligé d'effectuer des modifications.

PERCER LES SECRETS DU CODE

Même si tu as déjà utilisé des variables en **booléen** dans les chapitres précédents, je te recommande vivement de t'arrêter quelques secondes pour comprendre la manière dont la variable `anotherGo` a été utilisée dans le programme `duplicateur.py`. En voici les lignes le plus importantes :

```
anotherGo = True
while anotherGo:
    choix = menu()
    if choix == 8:
        anotherGo = False
```

En programmation, la structure de cette boucle est assez commune : elle se répète au moins une fois, te demande si tu veux engager un autre tour et, en cas de réponse négative, met fin à la boucle. Il existe plusieurs manières d'arriver au même résultat en Python, notamment en utilisant une variable booléenne, qui constitue une option idéale, car elle permet de voir clairement comment le programme fonctionne.

Python dispose d'une instruction nommée `break`, qui permet de briser une boucle. Même si tu ne vas pas l'utiliser dans ce livre, tu peux toujours effectuer une recherche sur Internet et t'amuser à modifier ta boucle en remplaçant la variable booléenne par cette instruction `break`.

Une **variable booléenne** est une variable qui ne peut prendre qu'une valeur sur deux, soit `True` ou `False`. Elles tiennent leur nom du mathématicien George Boole, qui a beaucoup travaillé sur la logique formelle dans les années 1980, une discipline qui a permis de poser les fondements de l'informatique moderne. Tu peux en lire davantage sur Boole en te rendant sur cette page : `http://fr.wikipedia.org/wiki/George_Boole`.

Afficher le menu

Ajouter un menu à ton programme peut s'avérer très utile s'il propose beaucoup d'options. Ce menu affiche toutes les options disponibles et effectue un tour de boucle, en attendant que tu saisisses le numéro de l'une d'elles. Si tu saisis un mauvais numéro, le menu s'affichera à nouveau en te demandant de réessayer.

1. Modifie la fonction du menu et remplace-la par les lignes de code suivantes (attention à l'indentation) :

```
def menu():
    while True:
        print("MENU DU DUPLICATEUR")
        print(" 1. CONSTRUIRE la salle de reproduction")
        print(" 2. AFFICHER les fichiers existants")
        print(" 3. SCANNER le contenu de la salle de  ↩
reproduction dans un fichier")
        print(" 4. CHARGER le contenu d'un fichier dans ↩
la salle de reproduction")
        print(" 5. IMPRIMER le contenu de la salle de ↩
reproduction en face de Steve")
        print(" 6. NETTOYER la salle de reproduction")
        print(" 7. DEMOLIR la salle de reproduction")
        print(" 8. FERMER")
        choix = int(raw_input(«choix : «))
        if choix < 1 or choix > 8:
            print("Tu dois choisir un chiffre entre 1 et 8")
        else:
            return choix
```

Enregistre ton programme et exécute-le. Qu'est-ce qui change par rapport au menu factice ? Sur la figure 6-8, tu peux voir ce à quoi ressemble le vrai menu.

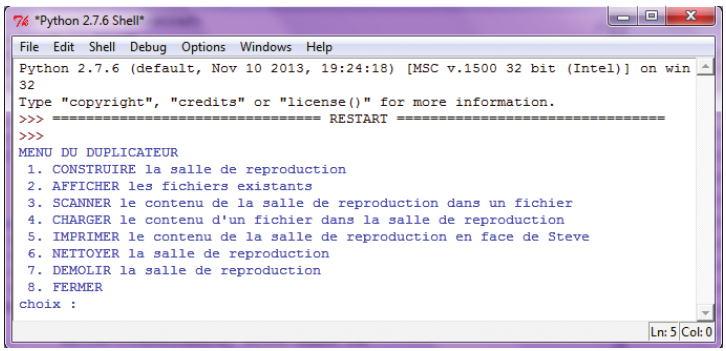

FIGURE 6-8 Le menu

En posant la structure de ton programme, avec son menu et toutes ses fonctions factices, puis en le complétant en détail fonction par fonction et en le testant, tu travailles exactement comme le ferait un ingénieur professionnel qui développe ses propres applications informatiques. La meilleure méthode pour concevoir un programme consiste en effet à l'écrire pas à pas, en testant tes changements au fur et à mesure, jusqu'à ce que tu en viennes à bout. De cette manière, tu pourras faire une démonstration à quiconque te le demandera en un rien de temps !

Bâtir la salle de reproduction

La salle de reproduction sera en verre et sera ouverte d'un côté pour que ton joueur puisse y entrer et facilement créer ou supprimer des blocs à l'intérieur.

Pour cela, tu dois remplacer la fonction `constructionSalle()` par les lignes de code suivantes. Fais attention aux lignes longues qui se trouvent dans la fonction `setBlocks()`. Tu remarqueras cependant que je n'ai pas utilisé la flèche ↲ dans les lignes suivantes, car ici Python me permet de les diviser en plusieurs lignes (pour en savoir plus sur cette exception, tu peux lire l'encadré « Percer les secrets du code ») :

```
def constructionSalle(x, y, z):
    global sallex, salley, sallez
    sallex = x
    salley = y
    sallez = z
    mc.setBlocks(sallex, salley, sallez,
                 sallex+DIMENSIONSX+2, salley+DIMENSIONSY+2,
                 sallez+DIMENSIONSZ+2, block.GLASS.id)
    mc.setBlocks(sallex+1, salley+1, sallez,
                 sallex+DIMENSIONSX+1, salley+DIMENSIONSY+1,
                 sallez+DIMENSIONSZ+1, block.AIR.id)
```

Sauvegarde ton programme et exécute-le à nouveau. Essaye l'option 1 du menu pour t'assurer que ton programme te permet bien de construire la salle de reproduction. Déplace-toi dans Minecraft et choisis l'option 1 à nouveau, puis regarde ce qu'il se passe ! La figure 6-9 te montre la salle de reproduction telle que tu devrais la voir.

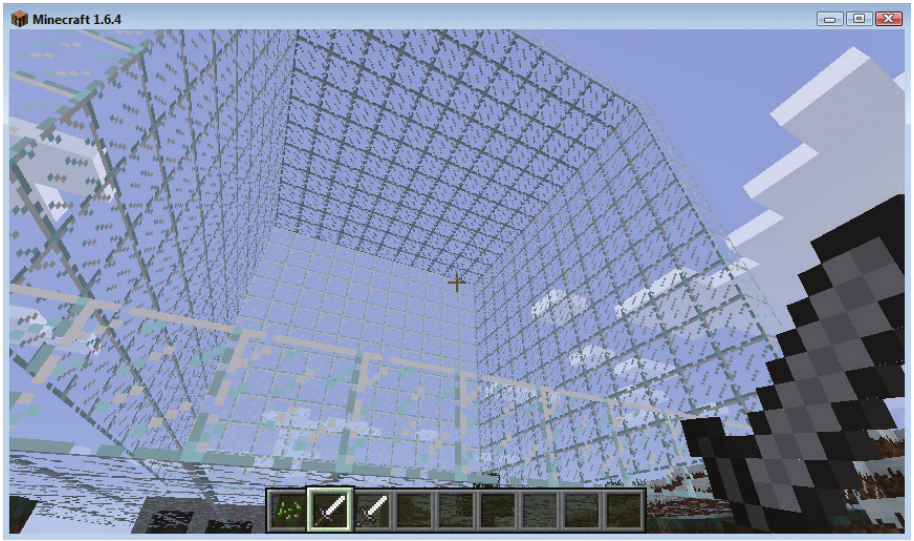

FIGURE 6-9 La salle de reproduction construite sous tes yeux

PERCER LES SECRETS DU CODE

Python utilise l'indentation (espace ou tabulation) au début des lignes de code pour distinguer les divers groupes d'instructions. Lorsque tu as utilisé des boucles, les mots-clés `if/else` ou même des fonctions, tu as indenté tes blocs de code en fonction. En programmation, cette pratique est peu commune et distingue le langage Python des autres langages de programmation comme le C ou le C++, qui utilisent des caractères spéciaux, tels que les accolades {}, pour regrouper des instructions. Tu dois donc faire particulièrement attention à l'indentation dans Python, sinon ton programme ne fonctionnera pas correctement !

La plupart du temps, Python ne te permet pas de séparer des lignes en plusieurs morceaux, ce qui explique pourquoi tu vois souvent la flèche ↲ dans ce livre, qui t'indique que la ligne ne doit pas être coupée.

Cependant, il existe deux autres méthodes pour traiter les lignes de code trop longues dans Python.

La première méthode consiste à utiliser un caractère de continuation de ligne : si le dernier caractère de ta ligne est une barre oblique inversée (\), cela veut dire que tu peux passer à une autre ligne.

```
a = 1
if a == 1 or \
a == 2:
    print("oui")
```

Quant à la deuxième méthode, elle consiste à couper une longue ligne de code là où il est évident qu'elle ne peut pas finir. Par exemple, lorsque tu définis des valeurs initiales dans une liste ou quand tu utilises une fonction, tu peux tout à fait couper tes lignes de code aux endroits où Python saura pertinemment qu'une suite est attendue. Ci-dessous, tu peux voir deux exemples probants qui t'aideront à comprendre comment Python te permet de telles coupes. Les crochets et parenthèses restés ouverts indiquent à Python que la ligne n'est pas terminée et qu'elle finira dès qu'ils se refermeront :

```
noms = ["David",
"Steve",
"Joan",
"Joanne"
]
mc.setBlocks(x1, y1, z1,
x2, y2, z2, block.AIR.id)
```

Démolir la salle de reproduction

Après avoir exécuté ton programme de duplicateur à plusieurs reprises, tu finiras certainement avec de nombreuses salles de reproduction éparpillées un peu partout dans Minecraft. Or à chaque fois, c'est la dernière que tu crées qui fonctionne réellement. Si tu continues ainsi, tu te trouveras bientôt avec tellement de salles que tu ne sauras plus où donner de la tête ! Pour remédier à ce problème, tu vas devoir ajouter une fonctionnalité à ton programme qui t'invitera à démolir ta salle de reproduction après t'en être servi afin d'éviter que ton monde ne croule sous les vieux duplicateurs.

Ce programme de démolition ressemble en tous points au programme `deblayage.py` du chapitre 3. Tout ce que tu as besoin de connaître, ce sont les coordonnées extérieures de ta salle. Et comme ta salle est plus grande d'un bloc par rapport à ton espace de reproduction, qui lui est défini par `DIMENSIONSX`, `DIMENSIONSY` et `DIMENSIONSZ`, il te suffira d'effectuer un simple calcul mathématique.

Modifie ta fonction `demolitionSalle()` pour qu'elle ressemble à ceci :

```
def demolitionSalle():
    mc.setBlocks(sallex, salley, sallez,
                 sallex+DIMENSIONSX+2, salley+DIMENSIONSY+2,
                 sallez+DIMENSIONSZ+2, block.AIR.id)
```

Enregistre ton programme et exécute-le à nouveau. Démolir ta salle de reproduction ne devrait plus te poser de problème. Choisis simplement l'option 1 de ton menu pour la construire, puis l'option 7 pour la démolir.

Scanner dans la salle de reproduction

Tu as déjà écrit cette partie du programme dans ton script `scannage3D.py`, donc tu peux t'en servir en y apportant de légères modifications.

1. Remplace la fonction `scannage3D()` par le code suivant. Ce code est presque identique à celui du programme `scannage3D.py`. Seules les lignes en gras y ont été ajoutées pour afficher la progression des opérations de scannage dans le chat de Minecraft. En effet, scanner une grande salle peut prendre beaucoup de temps, alors il est parfois bien pratique de connaître l'avancée de ton programme pendant son exécution. Pour gagner un peu de temps, copie et colle les lignes de code de ton programme précédent :

```
def scannage3D(nomdefichier, originex, originey, originez):
    f = open(nomdefichier, "w")
    f.write(str(DIMENSIONSX) + "," + str(DIMENSIONSY) + "," ↩
 + str(DIMENSIONSZ) + "\n")

    for y in range(DIMENSIONSY):
        mc.postToChat("scannage en cours :" + str(y))
        f.write("\n")
        for x in range(DIMENSIONSX):
            ligne = ""
            for z in range(DIMENSIONSZ):
                blockid = mc.getBlock(originex+x, ↩
 originey+y, originez+z)
                if ligne != "":
                    ligne = ligne + ","
                ligne = ligne + str(blockid)
            f.write(ligne + "\n")
    f.close()
```

2. Enregistre ton programme et teste-le à nouveau. Pour cela, rends-toi dans la salle de reproduction et construis quelque chose, puis choisis l'option 3 dans ton menu. Ouvre le fichier qui a été créé pour vérifier que l'objet a été scanné correctement. Sur la figure 6-10, tu peux voir une partie d'un fichier scanné. Tu remarqueras qu'il contient beaucoup de zéros. C'est parce que ta salle a également scanné tous les blocs `AIR`.

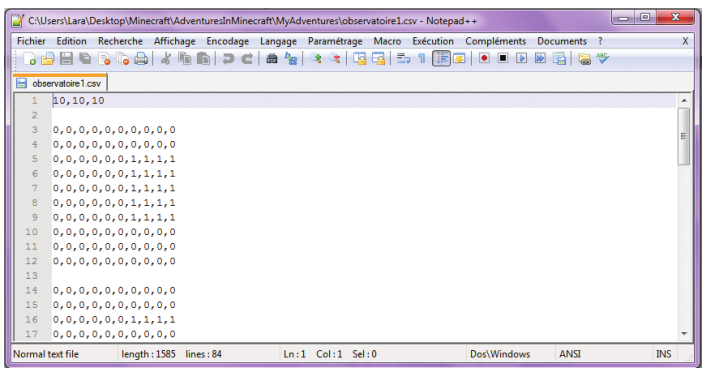

FIGURE 6-10 Le contenu d'un fichier scanné

Nettoyer la salle de reproduction

Besoin d'un nettoyage de printemps ? Il peut parfois s'avérer bien pratique de recommencer de zéro et de débarrasser la salle de tout ce qu'elle contient. Tu pourrais tout aussi bien parvenir au même résultat en la détruisant et en la reconstruisant, mais il est tellement simple d'ajouter une fonctionnalité de nettoyage qu'il serait dommage de s'en passer. Tout ce qui change par rapport à la fonction `demolitionSalle()`, ce sont les coordonnées.

1. Modifie la fonction `nettoyageSalle()` pour qu'elle ressemble au code suivant. Tu noteras que les coordonnées de départ sont toutes supérieures aux coordonnées des parois de la salle et que les coordonnées de fin ne sont pas plus élevées que celles de la fonction `demolitionSalle()`. De cette manière, tu vas nettoyer l'espace intérieur de la salle sans en détruire les murs :

```
def nettoyageSalle():
    mc.setBlocks(sallex+1, salley+1, sallez+1,
                 sallex+DIMENSIONSX+1, salley+DIMENSIONSY+1,
                 sallez+DIMENSIONSZ+1, block.AIR.id)
```

2. Sauvegarde ton programme et exécute-le à nouveau. Crée un objet dans la salle puis choisis l'option 6 pour savoir si tu es capable de nettoyer la salle en un claquement de doigts.

Imprimer dans la salle de reproduction

Imprimer (ou reproduire) un objet qui se trouve dans ta salle est une opération on ne peut plus simple à réaliser, car tu as déjà rédigé le programme qui te permet de le faire : il s'agit de ton programme `impression3D.py`. Le voici mis en contexte :

```
def impression3D(nomdefichier, originex, originey, originez):
    f = open(nomdefichier, "r")
    lignes = f.readlines()
    coords = lignes[0].split(",")
    dimensionsx = int(coords[0])
    dimensionsy = int(coords[1])
    dimensionsz = int(coords[2])
    idxligne = 1
    for y in range(dimensionsy):
        mc.postToChat("impression en cours:" + str(y))
        idxligne = idxligne + 1
        for x in range(dimensionsx):
            ligne = lignes[idxligne]
            idxligne = idxligne + 1
            donnee = ligne.split(",")
            for z in range(dimensionsz):
                blockid = int(donnee[z])
                mc.setBlock(originex+x, originey+y,
                            originez+z, blockid)
```

Enregistre ton programme et exécute-le à nouveau. Construis un objet dans la salle puis déplace-toi dans le jeu et choisis l'option 5 du menu. Tout ce qui se trouve dans ta salle sera scanné puis reproduit juste à côté de ton joueur. Tu peux répéter l'opération autant de fois que tu le souhaites et reproduire des objets un peu partout dans Minecraft. Essaye d'en reproduire dans le ciel et sous l'eau pour voir ce qu'il se passe ! Sur la figure 6-11, tu peux voir le résultat de l'opération après l'exécution du programme.

FIGURE 6-11 Un rocher reproduit à l'identique à côté du joueur

Créer un répertoire de fichiers

Ta dernière mission consiste à écrire une petite fonction bien pratique qui va te permettre de répertorier tous les fichiers dans un système de gestion de fichiers CSV. Tu pourrais aussi bien utiliser ton propre gestionnaire de fichiers pour cela, mais il est quand même plus intéressant d'ajouter cette fonctionnalité à ton programme, car tu auras ainsi tout ce dont tu as besoin dans un seul et même endroit :

1. Modifie la fonction `repertoireFichiers()` pour pouvoir utiliser la fonction `glob.glob()` qui lira tous les fichiers d'un répertoire et les imprimera. Pour en savoir plus sur cette fonction, réfère-toi à l'encadré « Percer les secrets du code ».

```
def repertoireFichiers():
    print("\nFICHIERS:")
    fichiers = glob.glob("*.csv")
    for nomdefichier in fichiers:
        print(nomdefichier)
    print("\n")
```

2. Enregistre ton programme et exécute-le à nouveau. Scanne quelques objets dans des fichiers CSV, choisis l'option 2 dans le menu et assure-toi qu'aucun objet ne manque. Sur la figure 6-12, tu peux voir les fichiers que j'ai créés lorsque j'ai exécuté ce programme sur mon ordinateur.

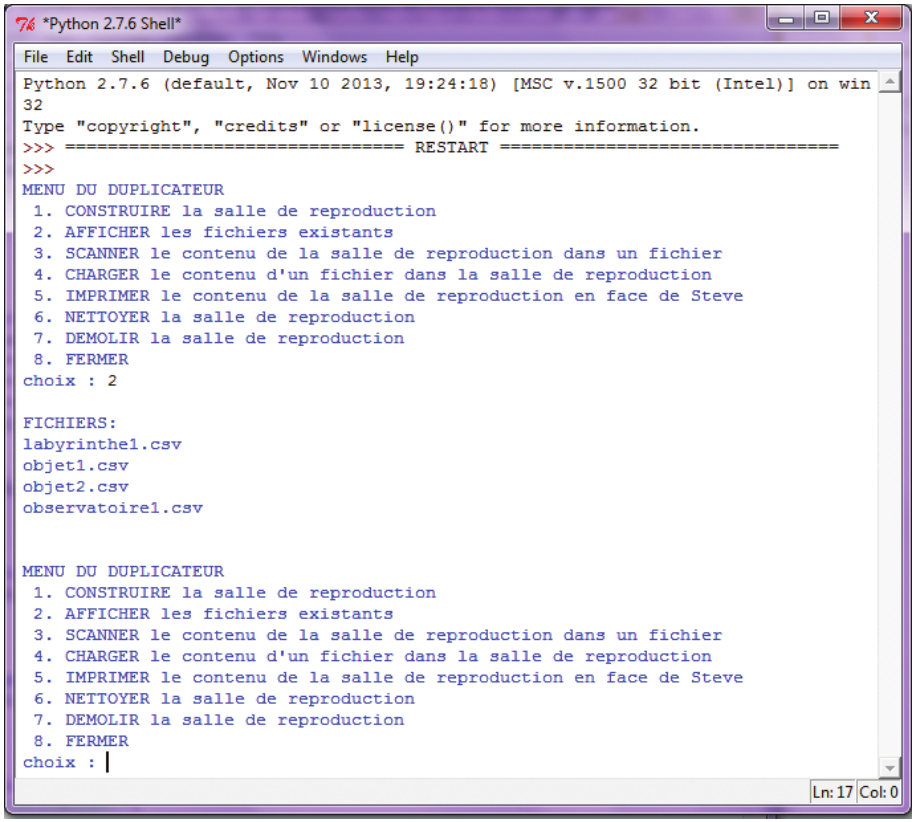

FIGURE 6-12 Le répertoire des fichiers CSV que tu as créés

PERCER LES SECRETS DU CODE

`glob.glob()` : quel curieux nom pour une fonction ! Si drôle qu'on le répète même deux fois !

Si le nom `glob` est utilisé deux fois, c'est parce que la première occurrence est le nom du module (le module `glob`) depuis lequel il a été importé au début du programme. La deuxième est le nom de la fonction `glob` qui se trouve à l'intérieur du module du même nom.

Mais à quoi sert la fonction `glob` et pourquoi porte-t-elle ce nom ?

Pour ce qui concerne le nom, il s'agit simplement de l'abréviation de l'expression « global command » (commande de base, en français), qui remonte aux tous débuts du système d'exploitation d'Unix. Pour en savoir plus sur l'histoire qui se cache derrière l'abréviation `glob`, tu peux lire l'article de l'encyclopédie Wikipédia qui y est consacré (en anglais) : `http://en.wikipedia.org/wiki/Glob_(programming)`.

Quant à son utilité, elle permet d'obtenir une liste de fichiers répondant à une recherche précise ou à un **métacaractère**. Par exemple, lorsque tu utilises `glob.glob("*.csv")`, Python analyse le répertoire dans le système de ton ordinateur et affiche une liste de tous les fichiers se terminant par `.csv`.

Donc, si tu as placé les fichiers `un.csv`, `deux.csv` et `trois.csv` dans ton dossier `MyAdventures` et que tu utilises `glob.glob("*.csv")`, tu obtiendras une liste Python qui ressemble à cela :

```
['un.csv', 'deux.csv', 'trois.csv']
```

Or, souviens-toi que la boucle *for* parcourt tous les éléments d'une liste. Donc, retiens bien les lignes de code suivantes, car elles sont très utiles :

```
for nom in glob.glob("*.csv") :
      print(nom)
```

Un **métacaractère** est un caractère spécial qui permet de sélectionner plusieurs noms ou mots identiques. Il fonctionne un peu comme un « joker » dans un jeu de cartes, car il peut représenter tout et n'importe quoi.

En Python, le métacaractère est souvent utilisé pour sélectionner plusieurs fichiers du même format dans un répertoire, par exemple tous les fichiers de type « CSV ». Pour cela, on utilise `glob.glob("*.csv")`. Dans cette commande, l'astérisque (`*`) indique la position du métacaractère et la fonction `glob.glob()` trouvera tous les fichiers `.csv` que contient l'ordinateur.

DÉFI

Si tu saisis un nom de fichier qui n'existe pas dans ton programme `duplicateur.py`, ce dernier va planter en t'indiquant une erreur et va laisser ta salle de reproduction telle quelle dans le monde de Minecraft, sans la détruire. Essaye d'améliorer ton programme en recherchant sur Internet un moyen de détecter les fichiers non existants pour éviter que ton programme ne plante.

Lorsque Martin a testé mon programme `duplicateur.py`, il y a trouvé un bug : la salle de reproduction est en effet plus grande qu'elle devrait l'être. Cela veut dire que si tu construis un objet de la taille de ta salle, une portion de blocs ne sera pas scannée dans le fichier. Donc ton objet sera reproduit sans cette portion de blocs. Cela pourrait devenir gênant, surtout si tu construis une maison ! Essaye de détecter la cause de ce bug et de voir si tu peux le réparer.

Tableau de références

Lire des lignes dans un fichier	Écrire des lignes dans un fichier

`f = open("données.txt", "r")`
`astuces = f.readlines()`
`f.close()`
`for t in astuces:`
`    print(t)`
	`f = open("scores.txt", "w")`
	`f.write("Victoria:26000\n")`
	`f.write("Amanda:10000\n")`
	`f.write("Ria:32768\n")`
	`f.close()`

Obtenir une liste de noms de fichiers analogues	**Enlever des espaces blancs en trop dans des chaînes de caractères**

`import glob`
`noms = glob.glob("*.csv")`
`for n in noms:`
`    print(n)`
	`a = "\n\n salut \n\n"`
	`a = a.strip()`
	`print(a)`

Explorer d'autres pistes avec les fichiers de données

Dans ce chapitre, tu as appris à lire et à écrire dans des fichiers de données. Cela t'ouvre d'infinies possibilités d'enregistrer et de restaurer des parties du monde de Minecraft et même d'introduire d'immenses volumes de données que tu peux trouver ailleurs, sur des sites Internet par exemple. Tu as aussi construit ton propre labyrinthe en 3D en le dotant d'innombrables couloirs sinueux et d'impasses. Puis tu as terminé en bâtissant une imprimante 3D en parfait état de fonctionnement, avec un menu très complet, qui va d'ailleurs te servir dans beaucoup de programmes !

- Il existe toute une panoplie de sites Internet recensant des « données en temps réel ». Lors de mes recherches pour la préparation de ce livre, je suis tombé sur un

site qui recense les données de stationnement de la ville de Nottingham : `http://data.nottinghamtravelwise.org.uk/parking.xml`. Ces données sont mises à jour toutes les cinq minutes et fournissent un suivi de toutes les entrées et sorties des aires de stationnement de la ville. J'ai écrit à l'occasion un programme Python qui traite ces données et reproduit des informations utiles, comme tu peux le voir sur ma page GitHub : `https://github.com/whaleygeek/pyparsers`. Essaye d'utiliser ce programme pour créer un jeu dans Minecraft avec des aires de stationnement et des voitures à l'intérieur. Tu pourrais par exemple en faire un jeu de chasse à la place de stationnement en temps limité !

- Tout ce que tu as toujours voulu savoir sur les labyrinthes en 3D est expliqué en détail sur ce site web : `http://www.astrolog.org/labyrnth/algrithm.htm`. Cette page inclut un tas d'exemples de labyrinthes en 3D multijoueurs, qui sont enregistrés dans des fichiers de traitement de texte ordinaires. Parcours un peu le site et vois si tu peux trouver un fichier de labyrinthe multijoueurs. Modifie ensuite ton programme de labyrinthe pour te servir des techniques apprises lors de la création de l'imprimante 3D et crée un immense labyrinthe composé de plusieurs couches. Si le cœur t'en dit, tu pourras même t'inspirer des quelques conseils du site pour écrire un programme capable de produire des labyrinthes automatiquement !

Niveau terminé : bravo ! Tu as réussi à briser les lois de la physique en faisant magiquement apparaître et disparaître d'immenses objets dans Minecraft, tout ça en un claquement de doigts !

Dans le prochain chapitre...

Dans le chapitre 7, tu vas apprendre à construire des objets complexes en 2D et en 3D à l'aide de Python. Tu apprendras notamment à contrôler la course folle du temps en construisant une horloge géante, à bâtir des polygones et autres objets complexes avec quelques lignes de code et à mener une expédition hors du commun sur les sentiers des pyramides !