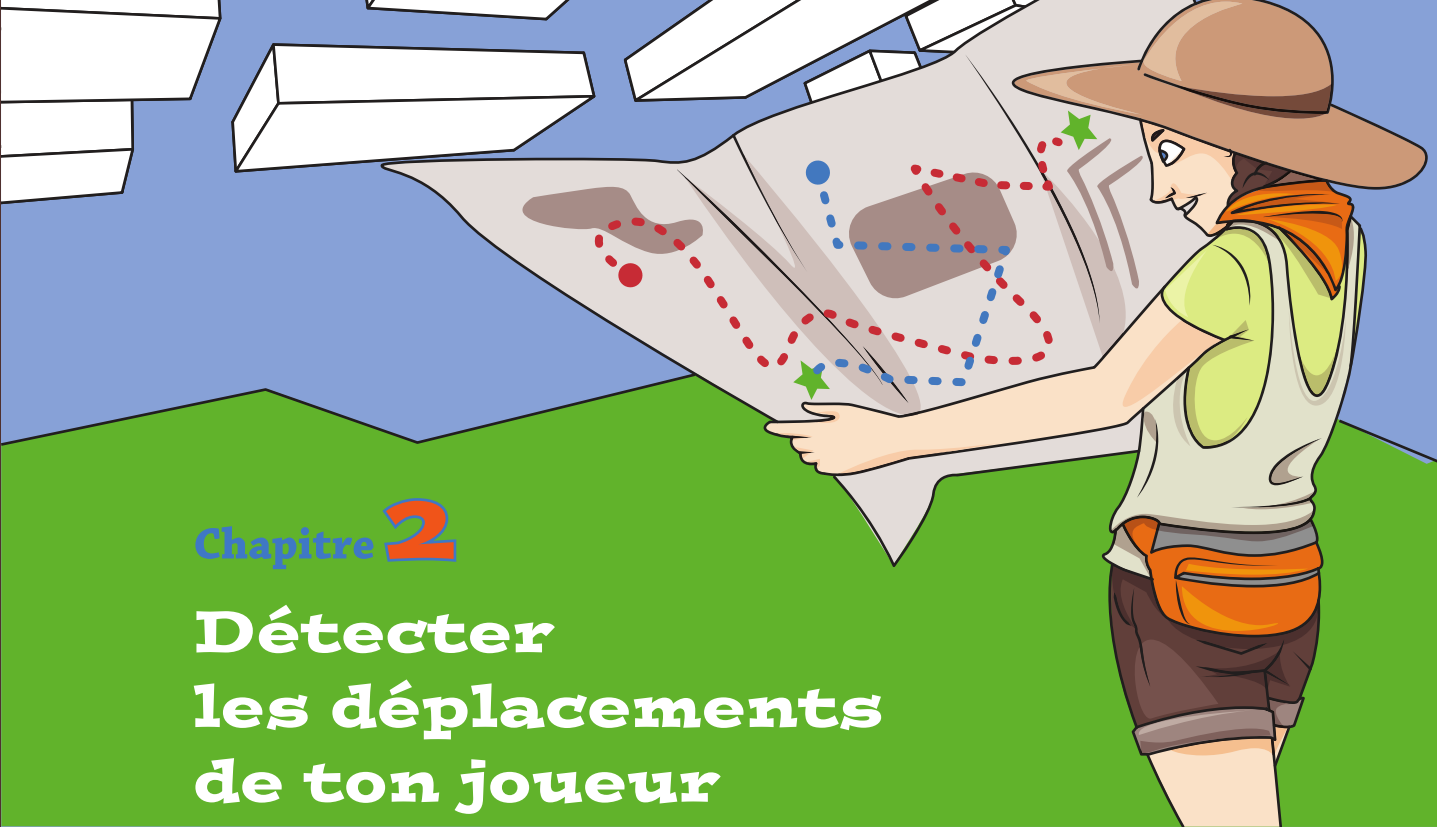

Chapitre 2

Détecter les déplacements de ton joueur

LORSQUE TU JOUES à Minecraft, tu découvres un monde que d'autres personnes ont conçu pour toi. Ce monde est amusant, mais il le serait bien plus encore si tu pouvais l'influencer à ta manière. Minecraft est fait pour ça. Le jeu t'offre l'environnement de programmation idéal pour laisser libre cours à ton imagination débordante et inventer des choses incroyables grâce au langage Python. Ce chapitre va te montrer comment profiter de cet environnement pour concevoir des choses amusantes en utilisant Python.

L'une d'elles consistera à créer un jeu dans le jeu, Minecraft étant l'univers que tu influenceras grâce à ton **programme**. En progressant, tu verras que la plupart des programmes Minecraft se composent de trois éléments principaux qui les rendent particulièrement amusants et captivants, à savoir : détecter des éléments du monde, par exemple la position de ton joueur, calculer des opérations, comme les résultats d'un jeu, et animer des choses, telles que les déplacements de ton joueur.

Dans ce chapitre, tu verras comment localiser ton joueur dans le monde et provoquer des actions de jeu en fonction de tes mouvements. Dans le jeu « Bienvenue à la maison », tu apprendras à créer un paillasson magique qui te souhaitera la bienvenue lorsque tu marcheras dessus. Nous t'initierons aussi à la technique du géorepérage en utilisant les blocs barrières de Minecraft. Et le jeu que tu vas créer te mettra au défi de récupérer des objets plus rapidement que tes amis. Enfin, tu apprendras à propulser ton joueur dans les airs si celui-ci reste bloqué dans un enclos trop longtemps !

Un **programme** est une série d'instructions que tu rédiges dans un langage informatique spécifique et que l'ordinateur respecte à la lettre. Les instructions varient selon le langage de programmation et tu dois t'assurer de les écrire correctement. Dans ce livre, nous utilisons le langage Python.

Une **instruction** est un terme informatique général qui signifie que tu commandes à ton ordinateur d'effectuer une action complète, en rédigeant par exemple une ligne de code comme `print("Bonjour Steve")`. Certains préfèrent employer le terme « commande », mais il est davantage utilisé pour désigner les ordres que tu saisis dans une invite de commandes pour que ton ordinateur exécute une action.

Le langage Python a sa propre définition du terme « instruction », mais pour les besoins de ce livre nous l'emploierons pour désigner un ordre ou une simple ligne de programmation écrite dans ce langage. Si tu souhaites en savoir plus sur le sujet, tu peux consulter la documentation en ligne en te rendant sur : `https://docs.python.org/2/`

Localiser ton joueur

Avant de commencer à localiser ton joueur, tu dois d'abord savoir comment le monde de Minecraft est conçu. Ce monde est uniquement composé de blocs, chacun étant un cube d'un mètre carré. Lorsque ton joueur, Steve, se déplace, le jeu connaît la position de ton joueur grâce à une série de **coordonnées**. Le monde est composé de blocs remplis de matériaux dont le volume correspond généralement à un mètre cube. Il existe quelques exceptions, comme les tapis et les liquides tels que l'eau ou la lave, qui occupent un volume plus petit. Mais ils ne peuvent remplir un volume inférieur à un mètre cube lorsqu'on les mélange à d'autres matériaux.

Une **coordonnée** est un ensemble de nombres qui indiquent une position. Dans Minecraft, les coordonnées 3D sont utilisées pour indiquer la position exacte de ton joueur dans le jeu en trois dimensions, chaque coordonnée contient donc trois nombres. Pour en savoir plus sur les données de localisation, tu peux consulter l'article `http://en.wikipedia.org/wiki/Coordinate_system`. Et si tu souhaites obtenir davantage d'informations sur la manière dont Minecraft traite ces données, rends-toi sur la page `http://minecraft.gamepedia.com/Coordinates`.

Connaître la position de ton joueur dans le monde de Minecraft te permet de provoquer des réactions intelligentes lorsque tu te déplaces. Par exemple, tu peux afficher des messages dans le chat à des endroits précis, bâtir des structures automatiquement lorsque tu es en mouvement et changer la position de ton joueur lorsque tu te rends dans certains endroits. Ce livre t'apprendra tous les secrets de ces formidables techniques de jeu, dont la plupart se trouvent précisément dans ce chapitre !

Nous appellerons ces coordonnées x, y et z. Tu dois faire très attention lorsque tu parles de coordonnées de position dans Minecraft, notamment si tu utilises des termes comme « gauche » et « droite », car la position des objets dépend de la direction vers laquelle tu te tournes. Il est préférable de penser ta position comme si tu disposais d'une boussole. La figure 2-1 montre comment les coordonnées x, y et z évoluent en fonction de tes déplacements dans le jeu.

Tu comprendras plus facilement comment tes coordonnées de position évoluent si tu les étudies directement dans Minecraft. Sur Raspberry Pi, tu peux voir tes coordonnées x, y et z s'afficher en haut à gauche de ton écran lorsque tu te déplaces. Sur PC ou Mac, il te suffit d'appuyer sur la touche `F3` pour les afficher. Pour en savoir plus sur les contrôles avancés du jeu, rends-toi sur la page `http://minecraft.gamepedia.com/Controls`.

- x augmente à mesure que tu te déplaces vers l'est et diminue lorsque tu vas vers l'ouest.
- y augmente lorsque tu te rapproches du ciel et diminue lorsque tu descends.
- z augmente à mesure que tu te déplaces vers le sud et diminue lorsque tu vas vers le nord.

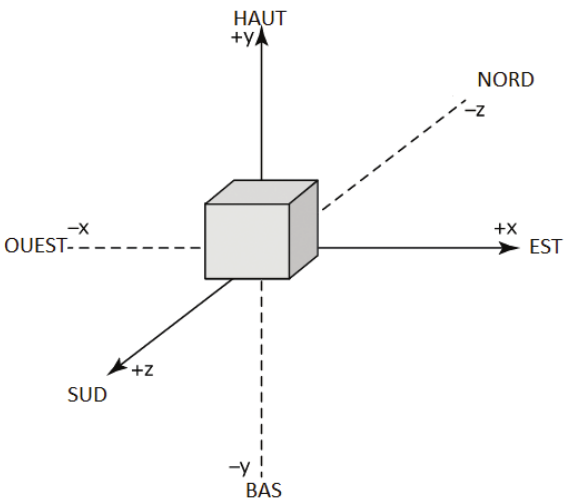

FIGURE 2-1 Les coordonnées x, y et z dans Minecraft correspondent aux pôles d'une boussole comme celle-ci.

Te préparer pour l'aventure

Maintenant que tu es parvenu à exécuter le programme « Bonjour le monde de Minecraft » dans le premier chapitre, il est temps que tu rédiges ton propre programme et que tu te mettes à jouer !

Dans le chapitre 1, nous t'avons expliqué comment faire fonctionner tous tes outils. À présent, tu vas perfectionner tout cela dans chaque nouveau chapitre. Toutes les étapes que tu dois connaître pour démarrer tes applications et commencer à rédiger des programmes sont décrites dans le chapitre 1, mais pour te rafraîchir la mémoire au début, nous effectuerons quelques rappels.

Si tu souhaites te rappeler comment faire fonctionner Minecraft sur ta machine, rends-toi sur le site Internet du livre : `www.editions-eyrolles.com/dl/14292` et regarde la vidéo intitulée « Adventure 1 » (en anglais).

1. 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.
2. Démarre IDLE, l'environnement de développement intégré (EDI) du langage Python, comme tu l'as fait dans le chapitre 1. C'est là que tu rédigeras et exécuteras tous tes programmes ; c'est en somme ta fenêtre sur l'univers de Python.

Afficher la position de ton joueur

Pour bien comprendre comment fonctionnent les coordonnées de localisation, le mieux est encore de rédiger un petit programme pratique. Celui que nous allons écrire dès maintenant permet d'afficher les coordonnées 3D de ton joueur à l'écran, même lorsqu'il est en mouvement.

1. Dans l'interpréteur de commandes Python, crée un nouveau programme via `File>New File`. Une nouvelle fenêtre va alors s'ouvrir dans laquelle tu vas rédiger le code de ton nouveau programme.
2. Avant de commencer à écrire un programme, nous te conseillons toujours d'enregistrer le fichier. Ainsi, si tu as besoin de quitter ton poste de travail, tu n'auras pas à lui trouver un nom à la hâte. Choisis donc de l'enregistrer, c'est ce que font tous les professionnels pour s'assurer de ne pas perdre leur travail. Pour cela, choisis `File>Save As` dans le menu de l'interpréteur de commandes Python, puis saisis le nom de ton programme : `ouSuisJe.py`. Assure-toi de bien enregistrer ton fichier dans le dossier `MyAdventures`.
3. Ton programme communiquera directement avec le jeu par le biais de l'interface de programmation (**API**) de Minecraft. Pour accéder à cette API, tu devras

importer le module *minecraft* qui permet à ton programme Python d'accéder à toutes les données du jeu. Utilise le code suivant :

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

4. Pour communiquer avec un jeu en cours d'exécution, tu dois t'y connecter. Pour cela, saisis la ligne suivante :

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

> Le langage Python est sensible à la casse. Prends donc garde à bien écrire les majuscules et les minuscules correctement ; c'est très important. Dans l'instruction que tu viens de saisir, le deuxième terme « Minecraft » doit commencer par une majuscule. Sans cela, l'instruction ne fonctionnera pas.
>
> 
>

5. Ensuite, demande à Minecraft de te fournir les données de position de ton joueur en utilisant `getTilePos()`, comme ceci :

```
pos = mc.player.getTilePos()
```

6. Enfin, demande-lui d'afficher les coordonnées de ton joueur. `print()` les affichera dans l'interpréteur de commandes Python lorsque tu exécuteras le programme :

```
print(pos.x)
print(pos.y)
print(pos.z)
```

7. Enregistre ton fichier en faisant `File>Save` dans le menu de l'éditeur de texte.
8. Puis exécute ton programme en cliquant sur `Run>Run Module`.

Tu devrais maintenant voir s'afficher les coordonnées de ton joueur dans l'interpréteur de commandes Python (voir la figure 2-2). Plus loin dans ce chapitre, tu utiliseras ces coordonnées pour connaître la position de ton joueur dans le monde et construire un paillasson magique qui te souhaitera la bienvenue lorsque tu marcheras dessus !

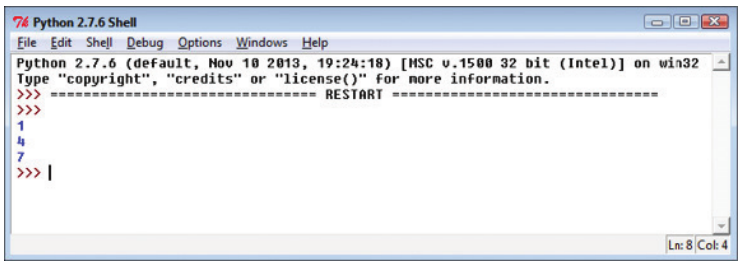

FIGURE 2-2 Utilise getTilePos pour afficher la position de ton joueur dans l'interpréteur de commandes Python.

L'acronyme **API** *(Application Programming Interface)* signifie « interface de programmation ». Une API te permet d'accéder en toute sécurité à certaines parties du programme d'une application depuis tes propres programmes. En l'occurrence, c'est en utilisant l'API de Minecraft que tu peux accéder au jeu dans tes programmes Python. Mais avant de connecter tes programmes Python au jeu via cette API, tu dois tout d'abord démarrer Minecraft. Le terme `mpci` dans la ligne `import mcpi.minecraft as minecraft` signifie *Minecraft Pi*, car la toute première version de l'API ne pouvait s'exécuter que sur Raspberry Pi.

Une **interface** est un ensemble de règles qui indiquent comment, en tant que programmeur, tu peux accéder à certaines parties du système d'exploitation d'un ordinateur. L'API est un ensemble de règles qui t'indiquent comment communiquer avec un programme en cours d'exécution – ici, le jeu Minecraft. Tous tes programmes Minecraft auront accès au jeu Minecraft en cours d'exécution via l'API de Minecraft.

Tous les programmes Python que tu vas créer devront accéder à l'API de Minecraft. Pour cela, tes programmes doivent être en mesure d'accéder au module *mcpi.minecraft* stocké dans le dossier `mcpi` de ton dossier `MyAdventures`. Pour garantir cet accès, tu dois toujours enregistrer tes programmes Python dans le dossier `MyAdventures`.

PERCER LES SECRETS DU CODE

Félicitations, tu viens d'utiliser une variable dans ton programme ! `pos.x` est une variable, au même titre que `pos.y` et `pos.z`.

Une **variable** est tout simplement un nom que tu donnes à un fragment de mémoire de ton ordinateur. Lorsque tu places un nom à gauche d'un signe égal, cela crée une variable, à laquelle tu affectes une valeur, comme dans :

```
a = 10
```

Si tu utilises une variable à l'intérieur d'une **instruction** `print`, le programme affichera la valeur de cette variable dans l'interpréteur de commandes Python. Donc, si *a = 10*, alors l'instruction `print(a)` affichera le nombre 10 – la valeur affectée à la variable `a` – lorsque tu y feras appel.

La variable `pos` que tu utilises dans `pos = mc.player.getTilePos()` est un type spécial de variable. Imagine qu'il s'agit simplement d'une boîte composée de trois compartiments nommés x, y et z : `print(pos.x)`

Déplace ton joueur à un autre endroit et exécute le programme à nouveau. Regarde maintenant les coordonnées qui s'affichent dans l'interpréteur de commandes Python ; celles-ci devraient avoir changé. Tu pourras utiliser ce petit programme quand tu le souhaiteras, notamment lorsque tu auras besoin de confirmer ta position dans le monde de Minecraft.

Une **variable** est une donnée que tu stockes dans la mémoire de ton ordinateur. Tu peux attribuer de nouvelles valeurs à tes variables n'importe où dans tes programmes. Tu peux également les rappeler et les afficher ou les utiliser dans des calculs. Pour te donner une meilleure idée de ce qu'est une variable, imagine qu'il s'agit de la touche mémoire d'une calculatrice.

Améliorer l'affichage de ta position

Avant de continuer, tu as la possibilité d'améliorer l'affichage de la position de ton joueur pour qu'elle soit plus facile à comprendre. En effet, placer les résultats sur trois lignes distinctes prend de la place. Il serait préférable de pouvoir les lire sur une seule et même ligne, comme ceci :

```
x=10 y=2 z=20
```

Pour cela, suis ces indications ci-dessous.

1. Modifie ton programme `ouSuisJe.py` en effaçant les trois instructions `print()` et en les remplaçant par cette ligne :

```
print("x="+str(pos.x) + "y="+str(pos.y) + "z="+str(pos.z))
```

2. Sauvegarde ton programme via `File>Save` dans le menu de l'éditeur, puis exécute-le en cliquant sur `Run>Run Module`. Regarde comment le résultat s'affiche. Il apparaît désormais sur une seule ligne. N'est-ce pas plus clair ?

PERCER LES SECRETS DU CODE

Dans la nouvelle version de ton programme `ouSuisje.py`, tu as réalisé un petit tour de magie que nous devons éclaircir ici.

En Python, l'instruction `print()` affiche tout ce que tu places entre les parenthèses et le résultat apparaît dans l'interpréteur de commandes Python. Tu as déjà affiché plusieurs informations à l'aide de cette instruction. Prends le temps de les examiner un peu, en commençant par cet exemple :

```
print("bonjour")
```

Comme tu l'as vu, `print()` affiche le terme « bonjour ». Les guillemets droits autour du mot indiquent à Python que tu souhaites afficher « bonjour » exactement comme tu l'as écrit. Si tu saisis des espaces entre les guillemets, elles apparaîtront aussi. Le texte qui se trouve à l'intérieur des guillemets s'appelle une **chaîne de caractères**.

Voici un autre exemple d'utilisation de `print()` :

```
print(pos.x)
```

Dans cette instruction, `print()` affiche la valeur affectée à la variable `pos.x`. Dans ce cas, il s'agit d'un nombre, mais celui-ci peut varier à chaque fois que tu utilises `print()`, en fonction de la valeur que tu auras affectée à la variable affichée.

Enfin, tu dois comprendre le rôle de `str()` dans ton code. L'instruction `print()` te permet de combiner des nombres et des chaînes de caractères de plusieurs manières. Lorsque tu rédiges des combinaisons de nombres et de chaînes dans Python, comme dans le code ci-dessous, tu dois indiquer à Python de convertir les nombres en chaînes pour pouvoir les unir aux autres caractères de la chaîne. `str()` est construite à l'intérieur de Python et convertit toutes les variables ou valeurs comprises entre ses parenthèses en chaîne de caractères. Le symbole plus (`+`) indique à Python d'unir « `x=`. » à toute valeur affectée à la variable `pos.x` pour créer une chaîne complète, qui s'affiche ensuite dans la fenêtre de l'interpréteur de commandes Python.

```
print("x= " + str(pos.x))
```

Maintenant, essaye de saisir cette même ligne sans `str()` et observe ce qui se passe :

```
print("x= " + pos.x)
```

Une **chaîne de caractères** est une variable qui permet de stocker toute une suite de lettres, de nombres et de symboles. En clair, c'est comme si tu enfilais des perles sur un collier ou un bracelet, chaque perle étant ornée d'un nombre, d'une lettre ou d'un symbole différent, le tout dans ordre bien défini et toujours identique.

Les chaînes de caractères sont utiles dans bien des cas, notamment si tu souhaites enregistrer ton nom, une adresse ou un message à afficher sur le chat de Minecraft.

Utiliser postToChat pour afficher ta position ailleurs

Avec le programme `ouSuisJe.py`, tu joues à Minecraft dans la fenêtre du jeu, mais ta position apparaît dans l'interpréteur de commandes Python, c'est assez gênant. Tu peux facilement y remédier en utilisant une technique que tu as déjà apprise dans le chapitre 1 : `postToChat` ! Voilà comment faire :

1. Remplace l'instruction `print()` par `mc.postToChat()`, comme ceci :

```
print("x=" + str(pos.x) + " y=" + str(pos.y) + " z=" +
      str(pos.z))
mc.postToChat("x=" + str(pos.x) + " y=" + str(pos.y) + " z="
      + str(pos.z))
```

2. Clique sur `File>Save` pour sauvegarder ton programme, puis exécute-le en cliquant sur `Run>Run Module` dans le menu de l'éditeur.

Félicitations ! La position de ton joueur s'affiche désormais dans le chat de Minecraft, ce qui doit te sembler bien plus pratique pour jouer.

Introduire une boucle de jeu

Tu en conviendras, exécuter ton programme `ouSuisJe.py` à chaque fois que tu souhaites savoir où tu es n'est pas très pratique. Heureusement, tu peux remédier à cela en modifiant ton programme et en y ajoutant une boucle de jeu. Presque tous les autres programmes que tu apprendras à rédiger dans ce livre incluent une boucle de jeu, c'est-à-dire un procédé qui leur permet d'interagir en continu avec Minecraft. De même, la plupart des jeux auxquels tu joues d'habitude continuent de tourner jusqu'à ce que tu les arrêtes. C'est une technique utile qu'il est bon d'acquérir. En informatique, on appelle cela une **boucle infinie**, car elle tourne sans jamais s'arrêter.

Une **boucle infinie** est une boucle qui ne s'arrête jamais. Elle tourne à l'infini et l'unique façon de l'arrêter est de mettre fin au programme Python lui-même. Pour cela, rends-toi dans l'interpréteur de commandes Python et clique sur `Shell>Restart Shell` dans le menu ou, lorsque tu te trouves dans l'interpréteur, maintiens les touches `Ctrl+F6` appuyées.

Pour ajouter la boucle, tu dois modifier le programme existant. Le nouveau programme sera très semblable, donc tu gagneras du temps. N'oublie pas de sauvegarder ton nouveau fichier sous un nom différent pour ne pas écraser le premier.

1. Commence tout d'abord par enregistrer ton nouveau fichier en sélectionnant `File>Save As` dans le menu de l'éditeur et appelle-le `ouSuisJe2.py`. Vérifie

qu'il est bien enregistré dans le dossier `MyAdventures`, sinon il ne fonctionnera pas.

2. Tu dois maintenant importer un nouveau module qui te permettra de ralentir le temps. En effet, si tu ne ralentis pas la boucle, ton chat sera inondé de messages qui t'empêcheront de voir ce que tu fais. Ajoute la ligne en **gras** ci-dessous au programme existant :

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

3. Modifie ensuite la partie principale de ton programme en ajoutant les lignes en gras suivantes, sans oublier d'indenter les lignes situées en dessous de `while True` pour indiquer à Python les instructions qu'il doit répéter. Réfère-toi à l'encadré « Percer les secrets du code » correspondant pour connaître la raison de cette indentation :

```
while True:
      time.sleep(1)
      pos = mc.player.getTilePos()
    mc.postToChat("x=" + str(pos.x) + " y=" + str(pos.y) ↩
+ " z=" + str(pos.z))
```

4. Va dans `File>Save` dans le menu de l'éditeur pour sauvegarder ton fichier.

5. Tu peux maintenant exécuter ton programme en sélectionnant `Run>Run Module` dans le menu de l'éditeur. Fais un tour dans le monde de Minecraft et tu verras tes coordonnées s'afficher dans le chat toutes les secondes.

PERCER LES SECRETS DU CODE

L'**indentation** est très importante dans tous les langages informatiques, car elle permet de mettre en évidence la structure des programmes et la manière dont s'imbriquent les blocs d'instructions. Mais elle est encore bien plus importante en Python, car le langage l'utilise pour comprendre la signification des programmes, contrairement aux autres langages, comme le C, qui utilise des crochets « {} » pour regrouper des instructions. Il faut notamment y faire attention lorsque tu utilises des boucles `while`, entre autres instructions, car elle indique quels blocs d'instructions appartiennent à la boucle et les répétera sans arrêt le cas échéant. Dans la boucle de jeu que tu as créée, toutes les instructions se trouvant en dessous de `while True:` sont indentées, car elles appartiennent à la boucle et seront répétées à chaque tour.

Dans l'exemple ci-dessous, tu peux remarquer que « bonjour » ne s'affiche qu'une seule fois, tandis que « tic » s'affiche chaque seconde. Quant aux instructions `time.sleep()` et `print()`, elles appartiennent à la boucle, car elles sont indentées :

```
print("bonjour")
  while True: # ceci est le début de la boucle
      time.sleep(1)
      print("tic")
```

Si en Python l'indentation importe davantage que dans les autres langages de programmation, c'est parce qu'elle permet d'indiquer, à l'aide d'alinéas, à quelles boucles les instructions appartiennent. Si tu te trompes d'un niveau par exemple, c'est tout le sens de ton programme que tu modifies.

L'**indentation** est l'espace qui se situe à gauche, en marge de tes lignes de programmation. Les espaces d'indentation permettent de structurer le programme et de regrouper plusieurs instructions en blocs distincts, sous des boucles ou d'autres instructions. En Python, les niveaux d'indentation sont importants, car ils permettent aussi de modifier le sens des programmes.

Il est important que tu comprennes le rôle de l'indentation. Lorsque tu indentes ton code, tu peux à la fois utiliser des espaces et des alinéas, mais lorsque tu combines ces deux modes d'indentation dans un seul et même programme, cela peut vite semer la pagaille. Il est donc préférable de ne suivre qu'une seule méthode d'indentation. La plupart des gens emploient la touche `Tab` pour indenter leur texte, car cela va beaucoup plus vite que d'utiliser des espaces. Si tu suis ce choix, alors tu devras t'y tenir le reste du temps et ne pas utiliser d'espaces.

Maintenant que tu es parvenu à exécuter une boucle de jeu infinie, ton programme ne s'arrêtera plus ! Avant d'exécuter ton prochain programme, arrête celui que tu viens d'exécuter via `Shell>Restart Shell` dans le menu de l'interpréteur de commandes ou maintiens les touches `Ctrl+F6` appuyées.

Bâtir le jeu Bienvenue à la maison

Il est temps de mettre en pratique cette technique de la boucle en développant une application simple. Le jeu « Bienvenue à la maison » détectera la position de ton joueur pour suivre ses déplacements dans le monde de Minecraft. Tu apprendras également à effectuer des repérages plus complexes à l'aide de l'instruction `if` en Python. Enfin, tu construiras un paillasson magique qui fera apparaître le message « Bienvenue à la maison » dans le chat dès lors que tu passeras dessus.

Tu peux regarder une vidéo didactique sur la création du jeu « Bienvenue à la maison » en te rendant sur le site Internet du livre `www.editions-eyrolles.com/dl/14292`. Choisis la vidéo « Adventure 2 » (en anglais).

Utiliser des instructions if pour créer un paillasson magique

Pour savoir si ton joueur se trouve sur le paillasson, tu utiliseras une instruction `if` qui te permettra de comparer la position de ton joueur avec celle du paillasson. Lorsque les deux positions seront identiques, cela voudra dire que ton joueur est arrivé à la maison.

Tout d'abord, pour observer comment l'instruction `if` fonctionne, fais l'essai par toi-même dans l'interpréteur de commandes Python.

1. Clique sur l'interpréteur de commandes Python. Veille à bien cliquer à droite des signes `>>>` pour t'assurer d'écrire ton code au bon endroit.
2. Saisis la ligne suivante dans l'interpréteur de commandes Python et celui-ci l'exécutera dès que tu appuieras sur la touche `Entrée`. Cela n'affichera rien, mais aura pour effet d'affecter le nombre `20` à la variable nommée `a` :

```
a = 20
```

3. Vérifie que la variable `a` contient la bonne valeur en saisissant l'instruction suivante dans l'interpréteur de commandes Python :

```
print(a)
```

   Le nombre `20` devrait apparaître à l'écran.

4. Essaye d'exécuter une instruction `if` pour voir si la valeur affectée à la variable `a` est supérieure à 10. Note que tu dois placer le deux-points à la fin de l'instruction `if`. Lorsque tu appuies sur `Entrée` à la fin de la première ligne, Python indente automatiquement la prochaine ligne pour toi :

```
if a>10:
    print("grand")
```

5. Maintenant, appuie sur `Entrée` à nouveau pour indiquer à l'interpréteur de commandes que tu as terminé de saisir l'instruction `if`. Tu devrais voir le terme « grand » s'afficher dans l'interpréteur de commandes. La figure 2-3 te montre ce à quoi ton écran doit ressembler.

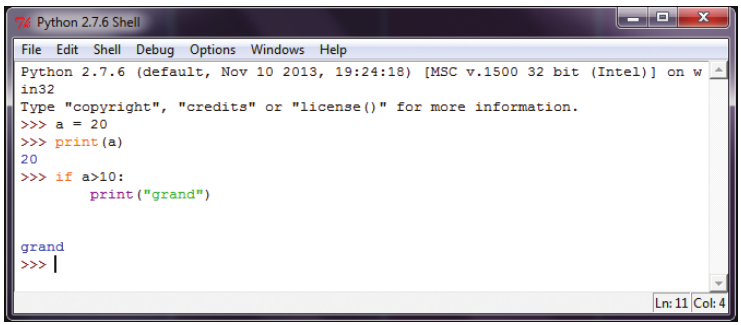

FIGURE 2-3 Utiliser une instruction if de façon interactive dans l'interpréteur de commandes Python

L'instruction `if` peut générer deux résultats. Par exemple, dans le code précédent, `if a>10` est soit `True` – c'est-à-dire « vrai » – (`a` est supérieur à `10`), soit `False` – « faux » – (`a` n'est pas supérieur à `10`). Les instructions `if` ne peuvent avoir que deux résultats possibles. Nous parlerons à nouveau des valeurs `True` et `False` un peu plus tard dans ce chapitre.

Connaître la position précise de ton joueur

Pour que ton programme puisse calculer si ton joueur franchit le paillasson magique, il devra vérifier au moins deux éléments de tes coordonnées. Si tu décèles que les coordonnées x et z du paillasson sont identiques à celles de ton joueur, tu pourras donc raisonnablement en conclure que ton joueur se trouve sur le paillasson. Fais-en l'expérience en utilisant le mot-clé `and` dans l'instruction `if` pour calculer les deux conditions en même temps. La coordonnée y importe moins ici. Nous n'allons donc pas la vérifier. Ainsi, même si ton joueur survole le paillasson, tu verras tout de même le message de bienvenue s'afficher.

Au préalable, effectue ces quelques étapes dans l'interpréteur de commandes Python pour t'assurer que tout fonctionne correctement.

1. Après les signes `>>>`, saisis les informations suivantes pour définir une variable :

```
a = 8
```

2. Saisis ensuite une instruction `if` contenant le terme `and`. Cette instruction sert à déterminer si la variable `a` se situe entre deux nombres. La première fois que tu saisiras ce code, tu verras le terme « entre » s'afficher dans l'interpréteur de commandes Python, car 8 est supérieur à 5 et inférieur à 10. Souviens-toi d'appuyer sur `Entrée` une deuxième fois pour mettre fin à l'indentation après le `if` :

```
if a>5 and a<10:
    print("entre")
```

3. Maintenant, modifie la valeur de `a` pour qu'elle soit inférieure à 5 et regarde ce qui se passe (voir la figure 2-4) :

```
a = 3
if a>5 and a<10:
    print("entre")
```

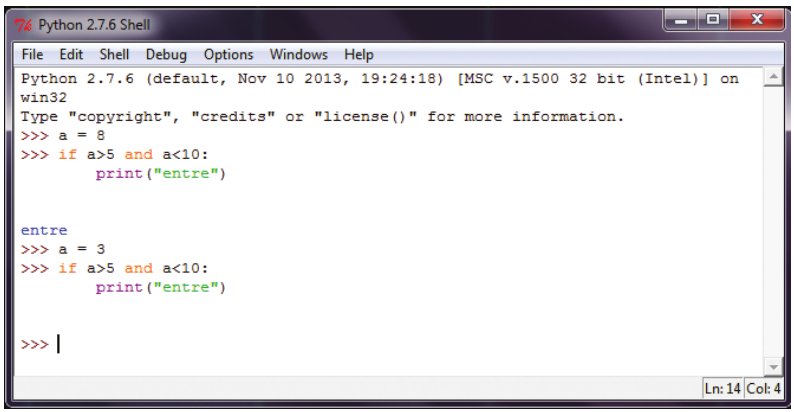

FIGURE 2-4 Tu utilises une instruction if avec and pour vérifier deux conditions.

DÉFI

Refais l'exercice précédent en employant des nombres différents, surtout des nombres situés entre 5 et 10. Avec quels nombres parviens-tu à afficher le terme « entre » ?

Lorsque tu testes des programmes, il est toujours très utile d'employer des nombres différents. Pour t'éviter un nombre incalculable de tests, contente-toi de vérifier ton instruction `if` uniquement avec des nombres avoisinant les conditions. Dans l'exemple précédent, il faudrait effectuer le test avec les nombres 4, 5, 6 et 9, 10, 11. Si les tests s'avèrent concluants, nous pourrons raisonnablement penser que tous les autres nombres fonctionnent aussi.

Construire un paillasson magique

Avant de rédiger ton jeu, tu dois tout d'abord bâtir une structure dans le monde de Minecraft pour donner à ton programme un contenu avec lequel il va interagir. Tout ce dont tu as besoin, en somme, c'est d'un paillasson. Mais avant tout, démarre Minecraft et charge le monde que tu as déjà visité.

1. Pour placer un paillasson sur le sol, choisis un objet de l'inventaire et effectue un clic droit pour le placer sur le sol devant toi. Un paillasson en bois fera parfaitement l'affaire.
2. Pour connaître les coordonnées de ton paillasson dans le jeu, exécute le programme `ouSuisJe.py` à nouveau et place-toi sur le paillasson. Note les valeurs des coordonnées x, y et z sur un papier, car tu en auras besoin au moment de rédiger ton prochain programme pour définir la position de ton paillasson dans le monde de Minecraft.

Tu dois probablement avoir hâte de commencer à construire d'immenses structures ! Tu apprendras à le faire automatiquement dans le chapitre 3, mais pour l'heure construis une maison simple pour te concentrer sur le programme. Crée par exemple une charmante maison en t'assurant de placer le paillasson à l'entrée, sur le pas de la porte.

Écrire le jeu Bienvenue à la maison

Maintenant que tu as compris comment l'instruction `if` fonctionne, tu peux te lancer dans la rédaction du jeu « Bienvenue à la maison » en suivant les indications ci-après.

1. Va dans `File>New File` dans le menu de l'interpréteur de commandes Python pour créer un nouveau programme.
2. Sélectionne `File>Save As` dans le menu de l'éditeur pour sauvegarder ton programme et nomme-le `bienvenueALaMaison.py`. N'oublie pas d'enregistrer ton programme dans le dossier `MyAdventures` pour qu'il fonctionne.
3. Importe les modules dont tu as besoin pour ce programme en insérant les lignes suivantes :

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

4. Connecte-toi au jeu Minecraft, en respectant bien la casse du mot `Minecraft` :

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

5. Introduis la boucle principale du jeu qui permet de détecter la position de ton joueur, en ajoutant du retard pour qu'elle ne s'exécute pas trop rapidement :

```
while True:
    time.sleep(1)
    pos = mc.player.getTilePos()
```

6. Ajoute l'instruction `if` qui permet de vérifier si ton joueur se trouve sur le paillasson, en utilisant `and` pour vérifier deux conditions. Les coordonnées x et z de ton paillasson doivent correspondre à la position de ton joueur pour que le programme puisse indiquer que celui-ci se trouve sur le paillasson. Tu as bien retenu les coordonnées de ton paillasson ? Écris-les dans ton code de cette manière pour que le programme puisse savoir où il se situe exactement :

```
    if pos.x == 10 and pos.z == 12:
        mc.postToChat("bienvenue à la maison")
```

Il est temps de voir si ton programme fonctionne ! Va dans `Run>Run Module` dans l'éditeur tout en te déplaçant dans Minecraft. Lorsque ton joueur franchit le pas de la porte, tu devrais voir apparaître « bienvenue à la maison », comme sur la figure 2-5. Super !

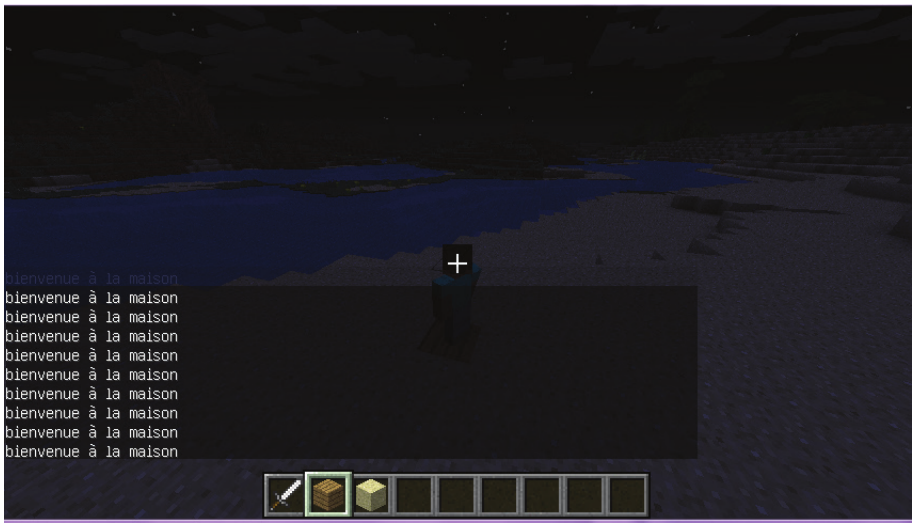

FIGURE 2-5 Lorsque tu franchis le pas de la porte, le message « bienvenue à la maison » s'affiche.

Dans l'instruction `if`, as-tu remarqué le double symbole `==` ? Une erreur fréquente consiste à n'utiliser qu'un seul symbole `=`. Essaye de reproduire l'erreur et vois ce qu'il se passe en exécutant ton programme. Python utilise le signe égal (`=`) une fois pour dire : « affecte la valeur située à droite du signe égal à la variable située à sa gauche ». Par exemple : `a = 3`. En revanche, Python utilise le signe égal deux fois (`==`) pour dire : « compare la valeur de droite avec la valeur de gauche ». Par exemple : `a == 3`.

Fais bien attention à l'indentation lorsque tu codes ton jeu « Bienvenue à la maison ». Les instructions incluses dans la boucle `while True` sont indentées une fois. L'instruction `mc.postToChat()` appartenant à l'instruction `if` est indentée deux fois. Si tu te trompes dans l'indentation, ton programme sera erroné.

Que tu le croies ou non, tu viens d'acquérir les connaissances fondamentales de programmation que tu utiliseras plus loin dans ce chapitre. Chaque programme Minecraft comprend les étapes suivantes : importer des modules, se connecter au jeu, former une boucle, détecter un événement et réagir en fonction de celui-ci. À partir de ces simples fondements, tu seras capable d'accomplir de grandes choses !

DÉFI

Penses-tu qu'en vérifiant la coordonnée y (celle de la hauteur), tu parviendras à mieux détecter la position de ton joueur par rapport au paillasson ? Fais-en l'essai. Modifie ton programme pour vérifier chacune des trois coordonnées du paillasson et regarde si cela fonctionne mieux.

POUR ALLER PLUS LOIN

À un moment donné, il t'arrivera probablement d'introduire des erreurs dans ton code sans le vouloir. Tu verras alors un message d'erreur s'afficher dans l'interpréteur de commandes Python. Nous te conseillons donc d'analyser d'un peu plus près le type d'erreurs que tu pourrais commettre, pour ne pas t'inquiéter lorsque tu te trouveras dans cette situation.

Il se peut par exemple que tu voies s'afficher une erreur de **syntaxe** si tu saisis mal un symbole ou si tu rédiges ton code de manière désordonnée. Tu obtiendras notamment un message d'erreur si tu omets un symbole ou si tu ajoutes une variable au mauvais endroit.

Par exemple, si tu saisis le signe égal seul dans ton interpréteur de commandes Python, tu verras s'afficher une erreur de syntaxe comme dans la figure 2-6, car ce signe est généralement utilisé pour des variables ou autres informations :

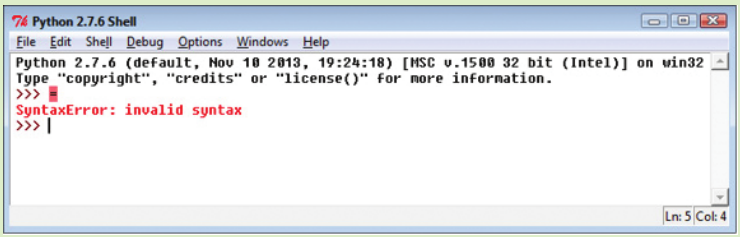

FIGURE 2-6 Python t'indique que ton code contient une erreur.

Maintenant, saisis la ligne suivante dans ton interpréteur de commandes Python :

```
print(zz)
```

Tu verras que Python te signale une erreur de nom, comme tu peux le voir sur la figure 2-7, car tu n'as pas encore affecté de valeur à la variable `zz`, donc Python n'est pas en mesure de reconnaître cette variable.

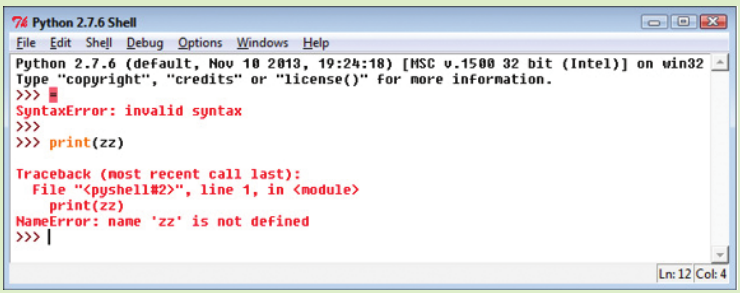

FIGURE 2-7 Python te signale une erreur de nom.

Essaye de saisir une boucle `while` en omettant volontairement d'insérer le deux-points (:) à la fin de la ligne :

```
while True
```

Là encore, tu obtiendras un message d'erreur de syntaxe, car tu n'as pas utilisé le deux-points alors que Python les exige.

Ne t'inquiète pas lorsque tu vois un message d'erreur dans ton programme. Analyse attentivement la ligne concernée et tâche de corriger l'erreur que tu as commise. Parfois, tu obtiendras un message d'erreur de syntaxe pour une erreur commise plus haut dans le code. Veille donc à bien analyser l'ensemble des lignes concernées par le message pour savoir où se situe le problème. Si tu n'y parviens pas, demande à un ami de le faire pour toi. Il est souvent bien plus facile de repérer les erreurs des autres ! Lorsque des erreurs sont signalées dans l'interpréteur de commandes Python, le numéro de la ligne où elles se trouvent est également indiqué. Pour le connaître, regarde en bas à droite de la fenêtre `IDLE` pour repérer la ligne où se situe ton curseur, ou rends-toi dans `Edit> GoTo` Line dans le menu `IDLE`.

La **syntaxe** fait référence aux règles d'un langage (en l'occurrence, le langage Python) et à l'ordre dans lequel tu saisis les informations.

Utiliser le géorepérage pour prélever un impôt

Tu vas maintenant rédiger un nouveau jeu qui s'appelle `impot.py` et qui te permet de construire un pré cerné d'une barrière. Ton programme aura la possibilité de détecter lorsque ton joueur entrera dans cet enclos et lui prélèvera une somme d'argent tout le temps qu'il passera à l'intérieur. Le défi que tu vas devoir relever consistera à retirer des objets de l'enclos le plus vite possible pour éviter de recevoir une facture trop salée.

Dans ton programme `bienvenueALaMaison.py`, tu as utilisé une instruction `if` pour vérifier si les coordonnées de ton joueur étaient identiques à celles du paillasson. Pour le savoir, tu dois placer ton joueur précisément au même emplacement que le paillasson. Or, il est possible de réaliser cette tâche encore plus rapidement grâce à la technique dite de **géorepérage**. C'est ce que tu vas faire ici dans ton nouveau jeu. Détecter la position de ton joueur sera d'autant plus simple que ce programme scannera une région du monde de Minecraft au lieu de se focaliser sur des coordonnées précises. Ainsi, tu pourras localiser ton joueur en scannant des régions entières et n'auras plus à te soucier de retrouver des points précis dans l'espace.

Plus généralement, le **géorepérage** est une technique qui consiste à encercler des coordonnées sur une carte. Lorsqu'un objet pénètre dans ce périmètre virtuel, quelque chose se produit en conséquence. L'objet en question peut être n'importe quoi. Il peut s'agir d'un joueur du monde de Minecraft ou, dans le monde réel, d'une personne, d'une machine comme une tondeuse à gazon ou encore d'un animal.

Dans bien des cas, la technique du géorepérage est utilisée pour avertir qu'un objet – un troupeau ou un véhicule par exemple – sort d'un périmètre défini. Pour en savoir plus sur l'utilisation du géorepérage en général, rends-toi sur la page `https://fr.wikipedia.org/wiki/Géorepérage`. Tu y découvriras comment le géorepérage est utilisé dans le monde réel, notamment pour suivre le trajet des éléphants dans la nature : `http://www.cbsnews.com/news/kenya-uses-text-messages-to-track-elephant/`

Pour que cette technique fonctionne, tu as besoin de deux choses : un objet ou une région à sonder et les coordonnées du contour de cet objet. Mais avant d'entrer dans le vif du sujet, construis un enclos à l'aide de barrières Minecraft et note les coordonnées de ses angles.

Tu n'es pas obligé de construire une barrière pour exécuter le programme, mais cela rendra le jeu un peu plus intéressant si tu le fais. En effet, une clôture t'empêche d'avancer, tu es obligé de sauter par-dessus ou de la longer pour trouver l'entrée de l'enclos. Or, ce sont les difficultés et les obstacles qui font tout l'attrait des jeux. Cette barrière Minecraft constituera les limites du périmètre de ton géorepérage virtuel et te permettra de mieux illustrer la question.

Ton enclos devrait ressembler à celui de la figure 2-8.

FIGURE 2-8 Un enclos de 10 blocs de largeur et 10 de longueur.

Enregistrer les coordonnées des angles de ton enclos

Pour pouvoir géorepérer ton enclos à l'aide de ton programme, tu as besoin d'enregistrer les coordonnées de ses angles en toute fiabilité afin de stocker les valeurs dans ton programme Python.

Le plus simple est encore d'exécuter ton programme `ouSuisJe.py` puis de te diriger vers les angles de ton enclos pour récupérer les coordonnées x, y et z de chacun des angles. Prends un papier et dessine un schéma de ton enclos en indiquant ces coordonnées, car tu en auras besoin lorsque tu rédigeras ton programme. Souviens-toi que la coordonnée z augmente à mesure que tu te déplaces vers le sud et diminue lorsque tu te diriges vers le nord, comme tu as pu le voir dans la figure 2-1.

Dans la figure 2-9, tu peux voir les coordonnées de l'enclos que nous avons bâti lorsque nous avons rédigé ce livre. Nous avons indiqué les coordonnées des quatre angles afin de pouvoir isoler les nombres les plus grands et les plus petits des directions x et z. Tu remarqueras que les coordonnées les plus petites se trouvent en haut à gauche.

Ton programme de géorepérage a besoin de quatre nombres. Tout d'abord, tu dois repérer les coordonnées x les plus grandes et les plus petites. Note-les ensuite sur ton schéma. Dans l'exemple de la figure 2-9, la coordonnée x la plus petite est de 10 et la plus grande est de 20. Nous avons choisi de nommer la coordonnée x la plus petite `X1` et la plus grande `X2`.

FIGURE 2-9 Maintenant que tu as noté les coordonnées des angles de ton enclos, tu es prêt pour la technique du géorepérage.

Selon la position de ton joueur dans le monde de Minecraft, il se peut que certaines coordonnées soient négatives (par exemple : `x=-5, y=0` et `z=-2`), ce qui peut compliquer un peu les calculs. Pour te faciliter la tâche, déplace-toi dans le monde de Minecraft afin d'obtenir des coordonnées positives (par exemple : `x=5`, `y=2`, `z=4`). Si tu disposes d'un Raspberry Pi, les coordonnées de ton joueur s'affichent en haut à gauche de ton écran de jeu. Si tu disposes d'un PC ou d'un Mac, appuie sur la touche `F3` pour les afficher.

Ensuite, tu devras calculer les plus petites et les plus grandes coordonnées `z`. Dans l'exemple de la figure 2-9, la coordonnée `z` la plus petite est de 10 et la plus grande est de 20. J'ai choisi de nommer la coordonnée `z` la plus petite `Z1` et la plus grande `Z2` :

```
X1 = 10
Z1 = 10
X2 = 20
Z2 = 20
```

Tâche d'attribuer le même nombre à tes coordonnées les plus grandes et les plus petites, car tu en auras besoin pour rédiger ton programme par la suite.

Écrire le programme de géorepérage

La dernière étape consiste à rédiger le programme qui va mettre la technique du géorepérage en pratique et prélever une somme d'argent dès lors que ton joueur entrera dans l'enclos. Ce programme est structuré de la même manière que le programme `ouSuisJe.py`.

Tu vas également utiliser ce qu'on appelle des **constantes** dans ce nouveau jeu. Tu pourras ainsi déplacer ton enclos plus facilement si besoin et n'auras pas à fouiller dans tes programmes à la recherche des valeurs à modifier pour réaliser cette action.

Comme pour une variable, une **constante** est le nom qu'on donne à un fragment de mémoire dans lequel tu peux stocker des valeurs qui restent identiques à chaque fois que le programme s'exécute.

Contrairement aux autres langages de programmation, Python ne dispose pas vraiment de manière de gérer les constantes. En revanche, pour pallier cela, les programmeurs Python utilisent généralement des conventions de programmation (c'est-à-dire une méthode). Ils écrivent notamment les constantes en lettres capitales pour pouvoir les reconnaître plus facilement et ainsi ne pas en modifier la valeur initiale. Les autres langages de programmation ont leur propre méthode pour reconnaître les constantes et dès lors qu'on essaye d'en changer la valeur, ils affichent un message d'erreur. Le langage Python ne dispose pas de cette particularité.

Passons maintenant à la rédaction du programme.

1. Ouvre tout d'abord une nouvelle fenêtre dans IDLE en sélectionnant `File>New File` dans le menu de l'interpréteur.
2. Choisis `File>Save As` dans le menu et nomme ton fichier `impot.py`.
3. Importe les modules dont tu as besoin pour ce programme :

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

4. Connecte-toi au jeu Minecraft :

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

5. Tu dois maintenant définir les valeurs des constantes de tes quatre points de géorepérage. Utilise les coordonnées que tu as calculées tout à l'heure. Pour ce qui nous concerne, nous avons repris les coordonnées de notre programme, mais il se peut que les vôtres soient différentes :

```
X1 = 10
Z1 = 10
```

```
X2 = 20
Z2 = 20
```

6. Crée ensuite une variable qui conservera un registre des impôts prélevés au joueur. Comme ton joueur n'aura pas encore subi de prélèvement d'impôt au début du jeu, tu dois créer une nouvelle variable que tu nommeras `impot` et que tu mettras à zéro :

```
impot = 0
```

7. Crée la boucle principale de ton jeu avec une instruction `while` pour faire tourner ton programme. Comme il va tourner sans arrêt, utilise l'instruction `sleep` pour le retarder d'une seconde entre chaque tour, cela te donnera en outre une idée du temps que ton joueur passe dans l'enclos. Plus tard, tu ajouteras `1` à la variable `impot` après chaque tour de boucle. Comme tu ralentis la boucle d'une seconde, cela veut dire que l'impôt augmentera de 1 toutes les secondes. Respecte bien l'indentation des lignes à partir de cette étape :

```
while True:
    time.sleep(1)
```

8. Pour savoir si ton joueur se trouve dans le périmètre de géorepérage, tu dois connaître sa position. Comme dans tes précédents programmes, utilise `player.getTilePos()` pour le savoir :

```
    pos = mc.player.getTilePos()
```

9. Tu es maintenant arrivé à l'étape la plus importante du programme. C'est ici que tu vas indiquer à ton programme d'utiliser les quatre coordonnées que tu as calculées plus tôt pour savoir si ton joueur se situe dans le périmètre et, si c'est le cas, pour lui prélever un impôt tout en le lui signalant dans le chat. Pour cela, saisis :

```
    if pos.x>X1 and pos.x<X2 and pos.z>Z1 and pos.z<Z2:
        impot = impot+1
        mc.postToChat("Tu dois la somme de :" + str(impot))
```

À ce stade, tu dois faire très attention à la manière dont tu indentes ton code. Python utilise l'indentation pour reconnaître les blocs d'instructions. Comme tu peux le remarquer, les instructions qui se trouvent en dessous de `while True:` sont indentées d'un cadratin. Cela permet à Python de savoir que ces instructions appartiennent à la boucle `while`. Mais, regarde l'instruction `if` de plus près : les deux lignes qui la suivent du programme sont indentées d'un cadratin supplémentaire. Cela veut dire qu'elles font partie de l'instruction `if` et qu'elles s'exécuteront uniquement si ton joueur se trouve dans l'enclos.

Sauvegarde ton programme à nouveau et exécute-le en sélectionnant `Run>Run Module` dans le menu de l'éditeur.

Amuse-toi à entrer et sortir de l'enclos avec ton joueur. Lorsque tu es dans l'enclos, tu dois voir apparaître toutes les secondes dans le chat un message t'indiquant le montant de l'impôt que tu dois payer. Lorsque tu en sors, le calcul de ton impôt s'arrête. Enfin, il ne t'aura certainement pas échappé qu'à chaque fois que tu retournes dans l'enclos, ton programme se souvient du montant que tu dois payer.

DÉFI

Pour que le jeu soit plus amusant, tu dois donner une mission à ton joueur. Crée quelques blocs à l'intérieur de l'enclos, puis fais fonctionner ton programme à nouveau. Le but est de récupérer tous les blocs en payant le moins d'impôts possible. Il te suffit de briser les blocs pour les récupérer. Même si ton programme ne reconnaît pas ces actions, cela rendra le jeu bien plus intéressant. Tu peux encore t'amuser à parsemer toutes sortes d'objets inventés dans ton enclos et inviter tes amis à relever le défi de les glaner en payant le moins d'impôts possible. Alors, à toi de jouer !

Déplacer ton joueur

Il existe un moyen de rendre ton jeu encore plus captivant en faisant notamment appel à une autre fonctionnalité de l'API de Minecraft que tu trouveras bien utile par la suite. Il s'agit de transporter ton joueur à un autre endroit dans Minecraft ! Pour cela, tu devras modifier ton programme `impot.py` et lui ajouter la possibilité d'éjecter ton joueur de l'enclos s'il y reste plus de trois secondes. Tu devras alors revenir dans l'enclos pour récupérer tes blocs.

Tu devras tout d'abord localiser un point en dehors de l'enclos où tu souhaites que ton joueur atterrisse. Le moyen le plus simple d'y parvenir est d'ajouter 2 points aux coordonnées X et Z les plus grandes de ton enclos pour obtenir d'autres coordonnées. Pour rajouter un effet spectaculaire à ton action, choisis une coordonnée Y à une altitude élevée. Le nombre que tu choisiras dépendra de l'altitude à laquelle tu as construit ton enclos, mais si ce dernier se trouve à une hauteur de `y = 0`, tu pourrais choisir par exemple une hauteur de 10. Ton joueur sera alors propulsé dans les airs et la gravité le fera retomber sur terre.

Tu vas donc modifier ton programme `impot.py` pour éjecter ton joueur hors de l'enclos s'il y reste plus de trois secondes, en suivant ces instructions.

1. Définis trois nouvelles constantes au début de ton programme pour indiquer la position d'un abri. Cet abri se trouvera juste en dehors de l'enclos et lorsque ton

joueur sera éjecté, il atterrira à cet endroit précis. Les nouvelles lignes à ajouter sont en gras :

```
X1 = 10
Z1 = 10
X2 = 20
Z2 = 20

ABRI_X = X2 + 2
ABRI_Y = 10
ABRI_Z = Z2 + 2
```

2. Tu vas maintenant préciser le temps que ton joueur peut passer dans l'enclos. Pour cela, tu dois créer une autre variable que tu peux appeler `dansEnclos`. Affecte-lui le nombre de secondes que ton joueur peut passer dans l'enclos. Elle va alors l'éjecter lorsqu'il aura passé trop de temps à l'intérieur. Là encore, n'ajoute que les lignes marquées en gras à ton programme :

```
impot = 0
dansEnclos = 0
```

3. Saisis ensuite une ligne de code pour incrémenter la variable `dansEnclos` de 1 lorsque ton joueur se trouve dans l'enclos. Pour cela, ajoute le code en gras :

```
if pos.x>X1 and pos.x<X2 and pos.z>Z1 and pos.z<Z2:
    impot = impot+1
    mc.postToChat("Tu dois la somme de: " + str(impot))
    dansEnclos = dansEnclos+1
```

4. Tu vas maintenant procéder à l'ajout d'une instruction `else` pour que le programme remette le compteur `dansEnclos` à zéro après que ton joueur a été éjecté en dehors l'enclos. Fais bien attention à indenter l'instruction `else` correctement pour que Python interprète bien ton code. Nous t'expliquerons un peu plus tard à quoi servent l'instruction `else` et le symbole `#` que tu vas utiliser. Pour l'instant contente-toi de saisir le code en gras :

```
    mc.postToChat("Tu dois la somme de:" + str(impot))
    dansEnclos = dansEnclos+1
else: # en dehors de l'enclos
    dansEnclos = 0
```

5. Enfin, ajoute quelques lignes de code supplémentaires à la fin de ton programme pour propulser le joueur dans les airs s'il reste dans l'enclos plus de trois secondes. La gravité le fera retomber sur terre et il pourra alors se ruer vers l'enclos à nouveau. Ce code doit se trouver tout à la fin de ton programme. L'instruction `if` doit être indentée d'un cadratin, car elle fait partie de la boucle `while True` et le bloc d'instructions qui suit `if` doit être indenté de deux cadratins, car il fait partie du `if`. Saisis le code suivant :

```
if dansEnclos>3:
    mc.postToChat("Tu es trop lent !")
    mc.player.setPos(ABRI_X, ABRI_Y, ABRI_Z)
```

6. Sauvegarde ton programme et exécute-le en cliquant sur `Run>Run Module` dans le menu de l'éditeur.

Tu vas bien t'amuser maintenant, car tu vas devoir redoubler d'ingéniosité pour entrer et sortir de l'enclos sans te faire éjecter ! Comme tout à l'heure, sélectionne une série d'objets à parsemer dans ton enclos et lance un défi à tes amis : qui de vous tous remportera cette folle course contre la montre en glanant le plus de blocs possible sans y laisser trop de plumes (voir la figure 2-10) ?

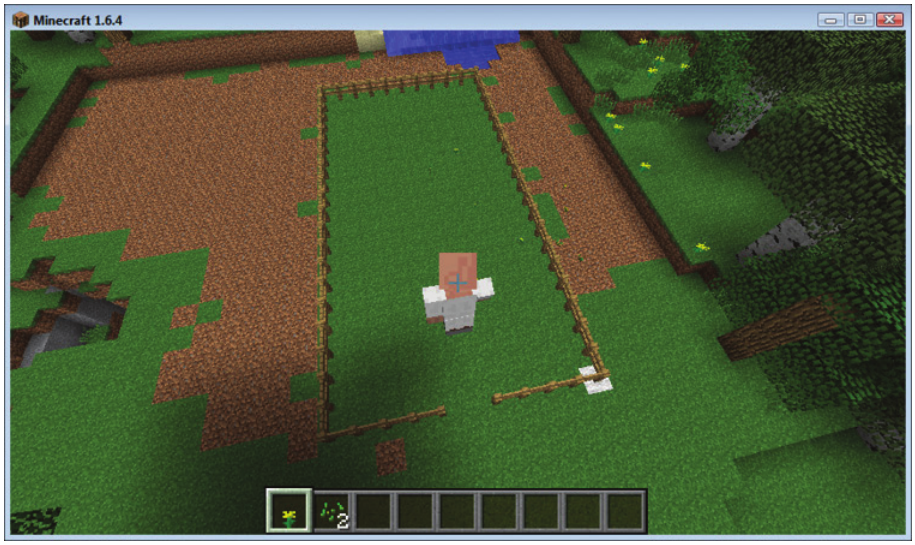

FIGURE 2-10 Ton joueur est propulsé dans les airs lorsqu'il reste trop longtemps dans l'enclos.

PERCER LES SECRETS DU CODE

Tu viens d'utiliser deux nouveaux éléments du langage Python. Il est temps de les approfondir un peu.

Tout d'abord, tu as utilisé l'instruction `else`, un mot-clé qui vient compléter l'instruction `if`. Il doit toujours se trouver au même niveau d'indentation que le `if` auquel il est associé et les instructions que tu souhaites rédiger sous `else` doivent aussi être indentées. En voici un exemple :

```
if a>3:
    print("grand")
```

```
else:
    print("petit")
```

Si la variable `a` renferme une valeur supérieure à 3, Python affichera « grand » dans l'interpréteur de commandes, sinon (`else`), c'est le terme « petit » qui s'affichera. Autrement dit, si `a>3` est vrai, alors le mot « grand » s'affichera, mais si `a>3` est faux, alors c'est « petit » qui s'affichera.

Une instruction `if` ne s'accompagne pas forcément d'une instruction `else` ; celle-ci est optionnelle. Mais tu auras parfois besoin de traiter la condition vraie différemment de la condition fausse.

Le deuxième élément que tu as utilisé est un commentaire. Tu verras qu'il est parfois bien utile de laisser un commentaire en marge de ton code pour te souvenir de certaines choses ou pour donner des indications à un autre lecteur.

Les commentaires commencent toujours par le symbole `#` :

```
# ceci est un commentaire
```

Tu peux laisser un commentaire n'importe où sur tes lignes de code. Dès lors que tu utiliseras le symbole dièse, tout ce qui suivra jusqu'à la fin de la ligne sera ignoré par Python.

Explorer d'autres pistes de localisation

Dans ce chapitre, tu as appris à mettre en pratique les fondements d'une boucle de jeu et à utiliser les techniques de localisation et de géorepérage pour créer un jeu amusant. Lorsque tu crées des jeux, nous te recommandons vivement d'inviter tes amis et ta famille à y jouer et de leur demander leur avis. C'est le meilleur moyen d'obtenir des conseils qui t'aideront à améliorer tes jeux.

- Le jeu « Bienvenue à la maison » est amusant, mais il a un problème : si ton joueur franchit le pas de la porte, il ne cesse d'afficher « Bienvenue à la maison » et cela t'empêche de bien voir. Peux-tu trouver un moyen de modifier le programme pour qu'il ne te salue qu'une seule fois lorsque tu entres chez toi ?
- Une bonne technique pour trouver de nouvelles idées est de découvrir d'autres jeux et d'observer comment ils tiennent les joueurs en haleine. Sois curieux, joue à d'autres jeux et fais une liste de toutes les caractéristiques que pourrait contenir un bon jeu, notamment en ce qui concerne le déplacement du joueur.
- Effectue des recherches sur Internet au sujet de la technique du géorepérage et vois comment l'industrie en fait usage. Si certaines idées t'intéressent, lance-toi dans l'écriture d'un programme Minecraft qui en reprend les principaux aspects, en utilisant les connaissances que tu as acquises dans cette aventure.

Tableau de références	
Importer l'API de Minecraft	**Se connecter à Minecraft**
`import mcpi.minecraft as minecraft`	`mc = minecraft.Minecraft.create()`
Obtenir la position du joueur	**Afficher un message dans le chat de Minecraft**

`pos = mc.player.getTilePos()`
`x = pos.x`
`y = pos.y`
`z = pos.z`
	`mc.postToChat("Bonjour Minecraft")`

Définir la position du joueur	

`x = 5`
`y = 3`
`z = 7`
`mc.player.setTilePos(x, y, z)`	

Niveau terminé : Tu as créé un super jeu qui réagit de plusieurs manières en fonction des déplacements de ton joueur dans Minecraft.

Dans le prochain chapitre...

Dans le chapitre 3, tu vas apprendre à construire de grandes structures, telles que des maisons, de façon automatique en utilisant des blocs et des boucles Python. Avec un seul programme Python, tu as la possibilité de bâtir d'immenses structures bien plus vite que si tu les construisais à la main. Tu pourras ainsi concevoir une ville entière en un rien de temps.