Langages et programmation

NSI - 1ère Activité
estimée : 4h

Contenu : Constructions élémentaires
Capacités attendues : Mettre en évidence un corpus de constructions élémentaires.

Le but de ces activités est de consolider les bases du langage Python à l'aide d'une interface 3D inspirée de Minecraft.

Cette interface repose sur le module *Ursina*, qui ajoute à Python quelques commandes utiles pour générer un "monde" en 3D.

I. Préparation

I.1. Arborescence des fichiers

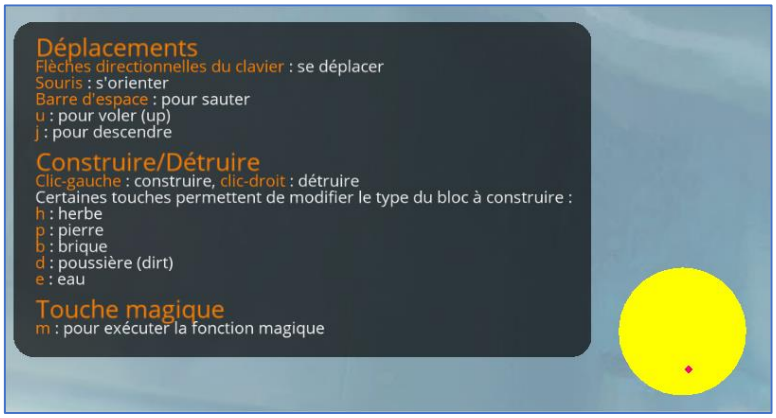
Si ce n'est pas encore fait, créer un dossier *NSI* dans vos documents puis mettre un sous-dossier *1NSI* et, à l'intérieur, un sous-dossier *Ursina*. Tous les fichiers que vous téléchargerez ou créérez seront à ranger dans ce dossier *TPUrsina*.

I.2. Installation d'Ursina

Lancer Thonny, et se rendre dans le menu **Outils → Gérer les paquets**. Taper "ursina" puis cliquer sur "**Rechercher**". Cliquer ensuite sur "**Installer**".

I.3. Test de l'installation

Télécharger le code à partir du site *Enseignement de l'informatique et du numérique au lycée Boissy d'Anglas*, l'ouvrir dans Thonny et l'exécuter par (**F5**).
On crée des blocs avec un clic-gauche de la souris et on en détruit avec un clic-droit. Pour se déplacer on utilisera les flèches directionnelles du clavier et la souris permet de diriger le regard.
Pour fermer le monde, taper sur la lettre q. Pour obtenir l'aide sur l'utilisation des touches, viser le soleil avec le pointeur de souris. La fenêtre d'aide s'ouvre alors :



Observer comme le code permettant de créer ce monde est relativement court !

I.4. Installation de l'API

Quelques fonctions très simples ont été rajoutées qui permettront d'utiliser plus simplement Ursina.
Télécharger le fichier ursacraft.zip et le dézipper dans un dossier de vos choix (clic-droit puis 'Extraire').
Ouvrir le fichier *ursacraft.py* dans Thonny et observer le code qui s'y trouve. Pas besoin de tout comprendre…
Quelques commentaires sont inclus pour donner des explications. Un commentaire en Python commence par #, il n'est pas *interprété* par Python, mais permet aux programmeurs de comprendre le code ou la partie du code s'y référant.
Chercher dans le code comment le joueur peut afficher l'aide pendant le jeu.
Tester cette API en exécutant le fichier (F5).
Faire en sorte d'afficher l'aide et d'utiliser les différentes commandes de jeu proposées dans cette aide.

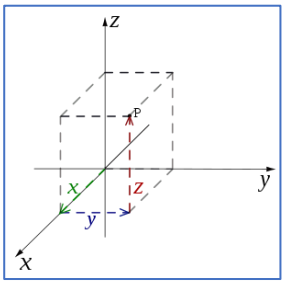
Commentaires sur quelques fonctions définies dans le script :

- la fonction magique permet de lancer une action lorsque le joueur tape sur la touche "m" ; pour l'instant elle ne fait rien ;
- la fonction *creer_monde* prend deux paramètres qui sont les dimensions du sol (de "l'aire de jeu") ; on peut les réduire ou les augmenter un peu mais pas trop.

Dans la suite, il faudra juste modifier le contenu de la fonction magique afin de réaliser les différentes actions proposées.

II. Création de blocs

II.1.



Observer le code de la fonction *magique* : la commande *pass* de Python est une instruction... qui ne fait rien !
Remplacer *pass* par construire_bloc(10, 10, 3) puis tester dans le jeu (rappel : touche *m*).
Voilà, la première pierre posée, ou du moins le premier bloc !
La fonction construire_bloc génère un seul bloc, repéré par ses trois coordonnées (*x*, *y*, *z*) :

- *x* et *y* sont les coordonnées "au sol" ;
- *z* est la hauteur du bloc.

II.2.

Modifier la hauteur du bloc (son *z*) puis tester. Modifier aussi les deux autres coordonnées et voir le nouvel emplacement du bloc.

II.3.

On va mettre un peu de hasard avec la fonction randint du module random (qui est déjà chargé par Ursina).
Par exemple :
random.randint(2, 10)
choisit un nombre entier au hasard entre 2 et 10 inclus.
Faire en sorte que la hauteur du bloc soit aléatoire.
Tester et appuyer plusieurs fois sur la touche "m".

II.4.

Deux variables ont été définies dans le script pour stocker les dimensions de l'aire de jeu : LONG et LARG.
Modifier l'emplacement du bloc afin qu'il se trouve n'importe où dans l'aire de jeu (et pas forcément au sol).

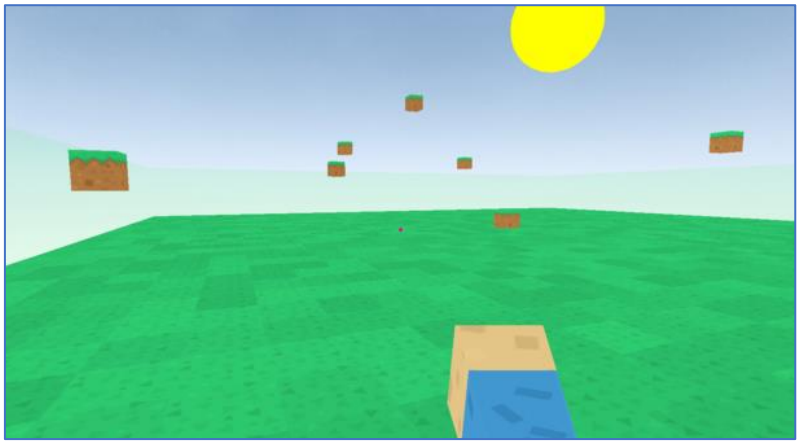


figure obtenue après plusieurs appuis sur la touche m

II.5.

La fonction construire_bloc a des paramètres optionnels, dont la texture du bloc (une texture est une image appliquée sur une surface).
Ainsi, l'instruction suivante permet de créer un bloc d'eau :

```
construire_bloc(20, 10, 3, texture = water_texture)
```

Modifier la fonction magique afin d'afficher deux blocs, un composé d'eau et un de pierre (*stone_texture*) n'importe où.

II.6.

La fonction construire_bloc a aussi le paramètre optionnel *couleur* :

```
construire_bloc(20, 10, 3, texture = None, couleur = color.red)
```

Dans cet exemple, la texture est à **None** (*None* = rien → pas de texture) pour que le bloc soit de couleur unie, mais on peut très bien mixer *texture* et *couleur* :

```
construire_bloc(20, 10, 3, texture = grass_texture, couleur = color.red)
```

Modifier la fonction magique afin d'afficher deux blocs, un bleu et un rouge n'importe où.

Recopier ici le code de la fonction magique :

III. Affichage de texte

On utilisera la fonction print de Python qui permet d'afficher du texte dans la console de Python.

III.1.

Mettre dans la fonction magique l'instruction print('Bonjour') et tester.
Testez ensuite avec print('45 divisé par 3 égal', 45/3). Ici deux choses sont affichées : un texte et un nombre.
Il est possible aussi d'afficher du texte dans la fenêtre de jeu, par exemple :
`Text(text = 'Hello les 1NSI !', position = (0.7, 0.2), background = True)`
(cette instruction n'est pas à connaître)

III.2.

Placer différents textes à différents endroits de l'écran, en modifiant les valeurs des paramètres (entre 0 et 1 pour ceux de position).
Remarque : la fonction print sert souvent au débogage d'un programme. Vous venez de voir que nous pouvons afficher des instructions dans un jeu sans avoir à utiliser celle-ci.

IV. Variables

IV.1.

Modifier la fonction magique ainsi :

```
def magique():
    hauteur = random.randint(0, 8)
    construire_bloc(random.randint(0, LONG), random.randint(0, LARG), hauteur)
    print('Bloc créé à la hauteur', hauteur)
```

puis tester en observant la console.
Nous avons ici utilisé une variable nommée *hauteur* pour stocker une valeur, à savoir la hauteur du bloc.

IV.2.

De même, modifier le code pour qu'il s'affiche dans la console : "Les coordonnées du bloc sont (..., ..., ...)" où les ... sont remplacés par les coordonnées aléatoires correspondantes.
Pour cela, on utilisera le formatage de texte :
L'opérateur **%** est utilisé pour formater un ensemble de variables contenues dans un *tuple*, ainsi qu'une chaîne qui contient du texte normal avec des symboles spéciaux comme **%s** (chaîne ou tout objet avec une représentation sous forme de chaîne), **%d** (entiers) ou encore **%f** (nombres à virgule flottante).
Exemple :

```
annee = 2022
evenement = 'Elections présidentielle'
print('Résultats année %d, %s' %(annee, evenement))

>>> 'Résultats année 2022, Elections présidentielle'
```

On peut aussi formater une chaîne en utilisant la méthode *format()*, voir ici ou encore ici.
Exemple :

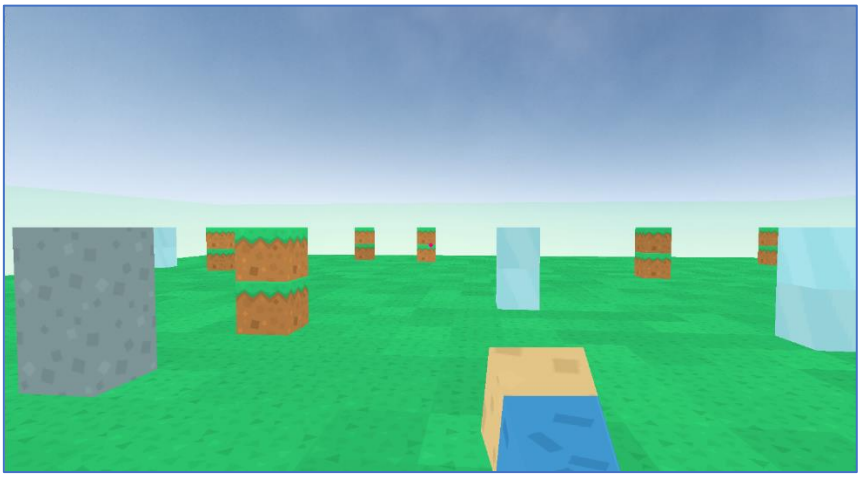
```
annee = 2022
event = 'Elections présidentielle'
print(f'Résultats année {year} {event}')

>>> 'Résultats année 2022 Elections présidentielle'
```

IV.3.

Modifier la fonction magique afin de créer à n'importe quel endroit **au sol** deux blocs superposés verticalement avec une texture aléatoire parmi l'herbe, la pierre, la brique, la poussière ou encore l'eau...
Il faut faire un choix parmi les 5 textures : créer une variable prenant comme valeur un entier entre 1 et 6 ;

Si la valeur trouvée aléatoirement vaut 1 **alors**
ma_texture = grass_texture
sinon si ... (elif ...)

Il est possible d'utiliser d'autres outils de Python mais nous verrons cela plus tard...



(figure obtenue après plusieurs appuis sur la touche m)

Recopier ici le code de la fonction magique :

V. Utilisation de boucles *for*

Revenons à nos constructions : comment faire par exemple une tour de 20 blocs de hauteur sur une base de 4 blocs de côté ?
Nous n'allons quand même pas devoir utiliser 4 × 4 × 20 = 320 instructions construire_bloc(...) dans lesquelles il faudrait choisir à chaque fois soigneusement les coordonnées de chaque bloc !
Heureusement, pour les tâches répétitives, nous disposons des boucles for, que nous associons ici avec la fonction range.
Ainsi :

```
for i in range(20) :
    instruction1
    instruction2
    ...
```

va exécuter 20 fois les instructions du bloc. Le bloc est constitué des instructions décalées vers la droite (indentation).
Il y aura ici 20 itérations de la boucle.
Enfin, la variable *i* aura successivement pour valeurs 0 ; 1 ; 2 ; ... ; **19** (rappel : pas 20 !).

V.1.

Essayer de deviner ce que va donner cette fonction puis vérifier:

```
def magique():
    for i in range(4):
        construire_bloc(10, 5, random.randint(0,20))
```

V.2.

Modifier la fonction magique pour qu'elle génère 50 blocs n'importe où dans l'aire de jeu (pas forcément au sol).

V.3.

Générer 10 "tours" de 3 blocs de haut, posées aléatoirement sur le sol.

Parfois le compteur de boucle n'est pas utilisé dans la boucle, par exemple :

```
for i in range(10):
    print("Bonjour !" )# ici i n'apparaît
```

mais parfois il l'est :

```
for i in range(10):
    print("7 fois ", i, "=", 7*i) # ici i est utilisée
```

Quand le compteur n'est pas utilisé dans la boucle, lui donner un nom ne semble pas utile.
C'est pourquoi, **en Python**, nous pouvons utiliser le symbole _ :

```
for _ in range(10):
    print("Bonjour !")
```

V.4.

Essayer de deviner ce que cette fonction va faire puis vérifier :

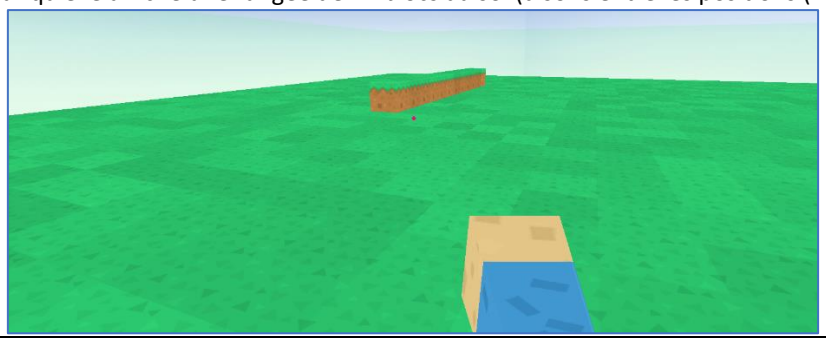
```
def magique() :
    for i in range(30) :
        construire_bloc(10, 5, i)
```

Remarque : le nom de la variable *i* utilisée pour la boucle peut être changé bien sûr. Il aurait été plus judicieux d'ailleurs d'appeler *hauteur* cette variable :

```
def magique() :
    for hauteur in range(30) :
        construire_bloc(10, 5, hauteur)
```

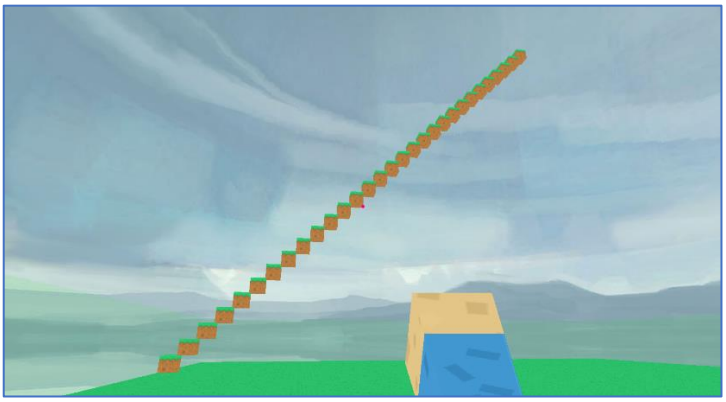
V.5.

Modifier la fonction pour qu'elle affiche une rangée de 12 blocs au sol (disons entre les positions (10, 10, 0) et (21, 10, 0)).



V.6.

Essayer maintenant de construire un "escalier" de 30 blocs, commençant à la position (10, 0, 0).



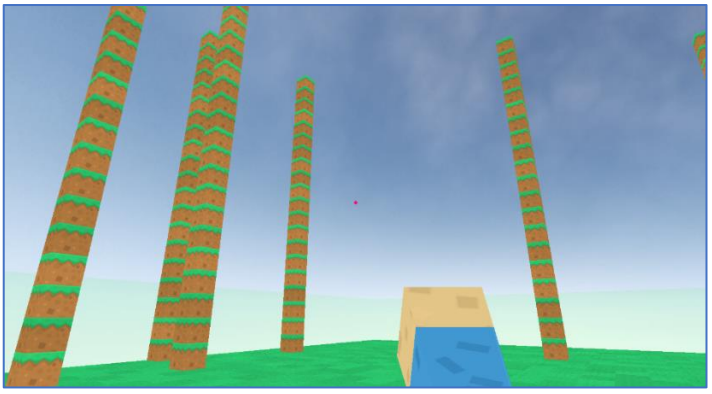
VI. Boucles *for* imbriquées

Il est parfois utile de placer une boucle dans une autre boucle, comme par exemple pour générer une table de multiplication :

```
for a in range(10):
    for b in range(10):
        print(a, " fois ", b, "égal", a*b)
```

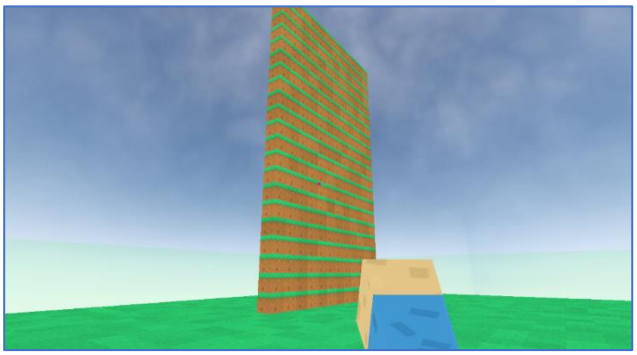
VI.1.

Modifier la fonction magique pour qu'elle affiche 10 "tours" de 15 blocs de hauteur, posées aléatoirement sur le sol.



VI.2.

Modifier la fonction magique pour qu'elle construise un mur de 12 blocs sur 20 de hauteur, placé au sol (disons entre les positions (10, 10, 0) et (21, 10, 0))



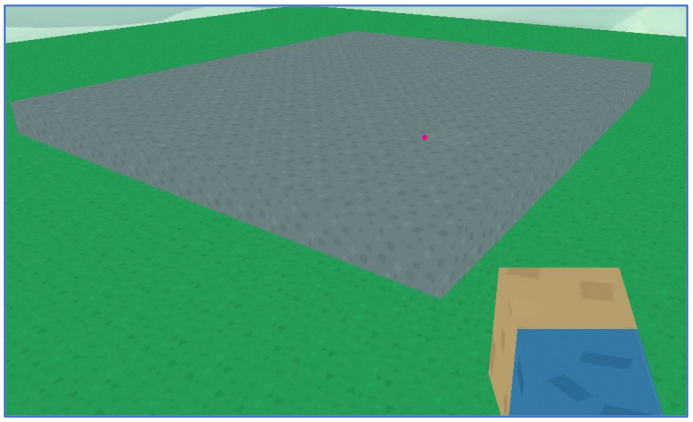
Aide :
Pour faire varier deux coordonnées, il faut deux boucles imbriquées.
Ici x doit aller de 10 à 21 et z doit aller de 0 à 20...
Au lieu de dessiner un segment (1 dimension), nous obtenons alors un rectangle (2 dimensions).

VI.3.

Même question mais avec un mur de 10 blocs de hauteur démarrant à une hauteur de 5 (avec x allant encore de 10 à 21).

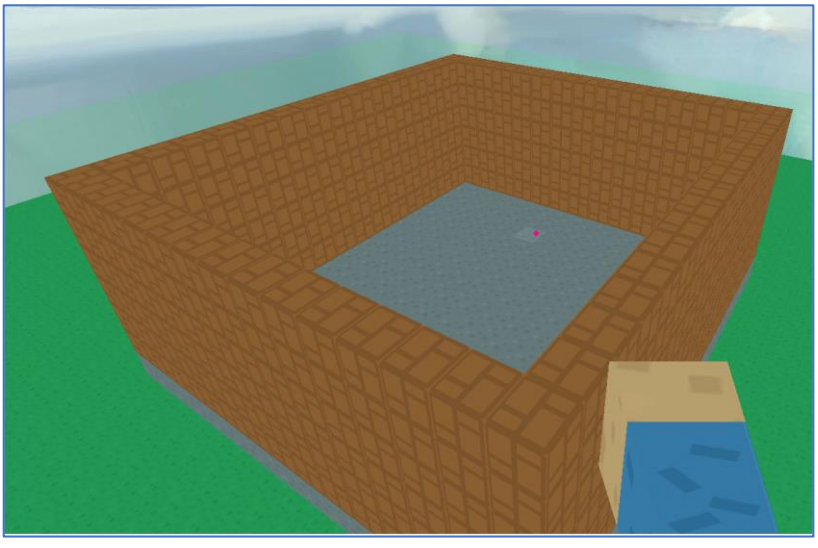
VI.4.

Votre fonction doit maintenant construire une *dalle* de blocs de pierre entre les positions (10, 7, 0) et (25, 22, 0).

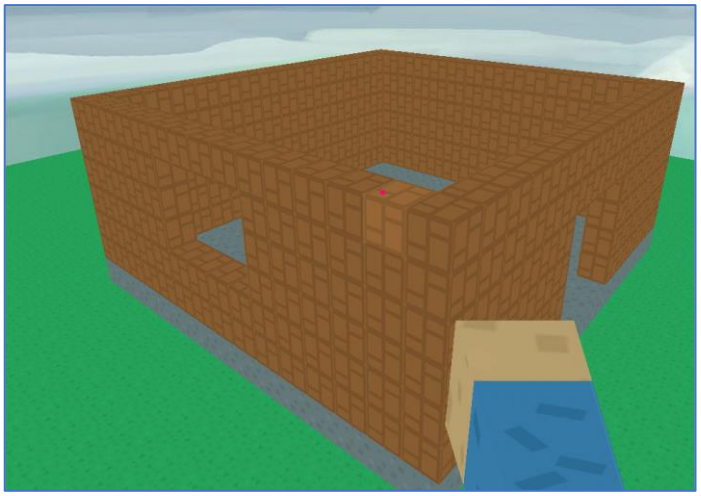


VI.5.

Il est temps de se mettre à l'abri : construire une maison de 6 blocs de hauteur posée au sol sur la dalle de blocs de pierre, entre des coins de coordonnées (10, 7, 0) et (25, 22, 0).



Prévoir une porte (4 blocs de haut et 4 de large, au milieu du mur) et une fenêtre (3 blocs de haut et 5 de large, au milieu du mur), en utilisant par exemple utiliser la fonction detruire_bloc.



VI.6. Éléments nécessaires pour la toiture.

Il est parfois utile qu'une boucle interne dépende de la boucle externe.
Par exemple, tester ceci :

```
def magique():
    for x in range(0, 7):
        for z in range(0, x + 1):
            construire_bloc(x, 12, z)
```

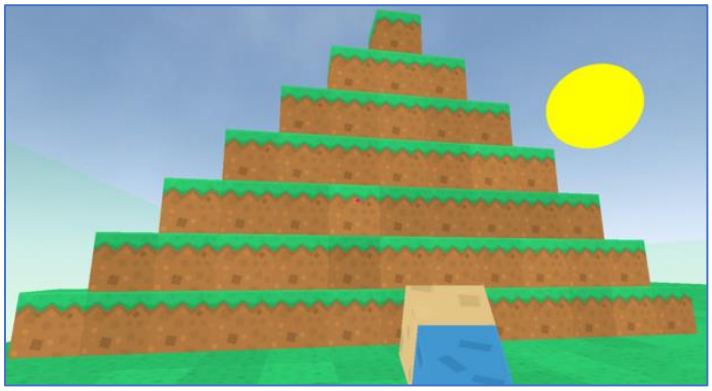
Ici, *x* varie de 0 à 6 pendant que *z* varie de 0 à *x* donc :

- pour *x = 0* : *z* varie de 0 à 0 donc un seul bloc à la hauteur 0 ;
- pour *x = 1* : *z* varie de 0 à 1 donc deux blocs aux hauteurs 0 et 1 ;
- pour *x = 2* : *z* varie de 0 à 2 donc trois blocs aux hauteurs 0 ; 1 et 2 ;
- pour *x = 3* : *z* varie de 0 à 3 donc quatre blocs aux hauteurs 0 ; 1 ; 2 et 3 ;
- etc.

d'où une construction "en escalier plein".

VI.7.

Essayer de compléter cet escalier de la façon suivante :



Aide

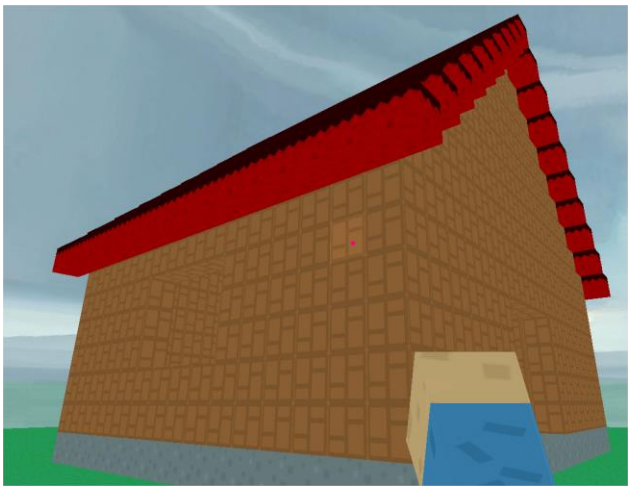
Conserver le code précédent pour la moitié de l'escalier. Dupliquer ce code pour l'autre moitié, modifiez les valeurs de *x*. Il reste à trouver comment calculer *z* en fonction de *x* :

- pour *x = 7* : *z* varie de 0 à 5 ;
- pour *x = 8* : *z* varie de 0 à 4 ;
- pour *x = 9* : *z* varie de 0 à 3 ;
- pour *x = 10* : *z* varie de 0 à 2 ;
- etc.
- donc *z* varie de 0 à ... (à vous de trouver !)

On pourrait également changer l'ordre des boucles x et z…

VI.8.

Reprendre le code de votre maison et lui ajouter une toiture rouge… Celle-ci doit comporter 2 pignons en escalier (en brique), et la couverture en blocs rouges. Pour la toiture, on la créera par ligne de blocs rouges, par l'intermédiaire de boucles *for*… On appliquera un bloc de déport par rapport au mur sur tout le pourtour.



Création pignons	**Création toiture**

VII. Fonctions

Quand une action est réalisée par plusieurs instructions, il peut être intéressant de créer une fonction.

VII.1.

Copier la fonction suivante dans le fichier ursacraft.py, avant la fonction magique :

```
def fleur(x, y, taille):
    construire_bloc(x, y, 0, cote = taille, texture = None, couleur = color.green)
    construire_bloc(x+0.25*taille, y, taille, texture = None, cote = taille, couleur = color.green)
    construire_bloc(x, y, 2*taille, texture = None, cote = taille, couleur = color.green)
    construire_bloc(x+0.25*taille, y, 3*taille, texture = None, cote = taille, couleur = color.red)
    construire_bloc(x+0.25*taille, y, 5*taille, texture = None, cote = taille, couleur = color.red)
    construire_bloc(x-0.75*taille, y, 4*taille, texture = None, cote = taille, couleur = color.red)
    construire_bloc(x+1.25*taille, y, 4*taille, texture = None, cote = taille, couleur = color.red)
    construire_bloc(x+0.25*taille, y, 4*taille, texture = None, cote = taille, couleur = color.yellow)
```

puis la tester en plaçant dans la fonction magique un appel de la fonction fleur avec comme arguments sa position et sa taille.

VII.2.

Faire en sorte que la fonction magique affiche plusieurs fleurs de positions et de tailles différentes.